VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Využití Windows Workflow Frameworku při vývoji aplikací

Windows Workflow Framework for Application Development

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jaroslav Dyba**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Využití Windows Workflow frameworku při vývoji aplikací
Windows Workflow Framework for Application Development

Zásady pro vypracování:

Windows Workflow Foundation je jedna z částí .NET framworku určená k modelování bussines procesů. Hlavní výhoda WF je v tom, že neposkytuje jenom grafický náhled na daný bussines proces, ale i možnost realizace procesu pomocí programového kódu. Při změně business procesu se mění jen náhled na proces.

Zásady pro vypracování:
1. Seznámení se s workflow.
2. Základní možnosti WF 4.0.
3. Případová studie zvoleného bussines procesu.
4. Implementace případové studie (vlastní aktivity).

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Kožusznik, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě, dne 3.5.2012 .....

Abstrakt

Tato diplomová práce se zabývá využití Windows Workflow Foundation 4.0 v procesu vývoje softwaru. Podíváme se, co nám tato nová technologie přináší, jak nám může pomoci. V úvodu se zamyslíme nad myšlenkou znovupoužitelnosti softwarových komponent, kterou tato technologie dále rozvíjí. Vysvětlíme si, co pojem workflow znamená a jaké typy workflow rozlišujeme. Krátce si představíme techniky bussines modelování a ukážeme si, jak jednotlivé modely implementovat s pomocí této technologie. Závěrem si ukážeme praktickou ukázku implementace vybrané části podnikového procesu pomocí Windows Workflow Foundation 4.0.

Klíčová slova

Windows Workflow Foundation, podnikový proces, Workflow, business modelování.

Abstract

This thesis deals with the use of Windows Workflow Foundation 4.0 during software development process. Let's see what this technology brings new and how it can help us with software construction. In the beginning we revise idea of reusability of software components which this technology continues to develop. We try to explain what the term “Workflow” really means and which types of workflow we distinguish. Shortly we will introduce the techniques of business process modeling and we show how to implement the business models using this technology. Finally, we present a practical example of implementation of selected business process using Windows Workflow Foundation 4.0.

Key Words

Windows Workflow Foundation, business process, Workflow, business modeling.

Seznam použitých zkratek

WF 4.0	Window Workflow Foundation 4
IS	Informační systém
ERP	Enterprise resouce plannig
WfM	Workflow management
C/E	Condition/Event
VS	Visual Studio
CID	CID International a.s.
WCF	Windows Communication Foundation
CRM	Customer relationship management
LIFO	Last In, First Out
FIFO	First In, First Out
FEFO	First Expirate, First Out
TFS	Team Foundation Server
MSBuild	Microsoft Build

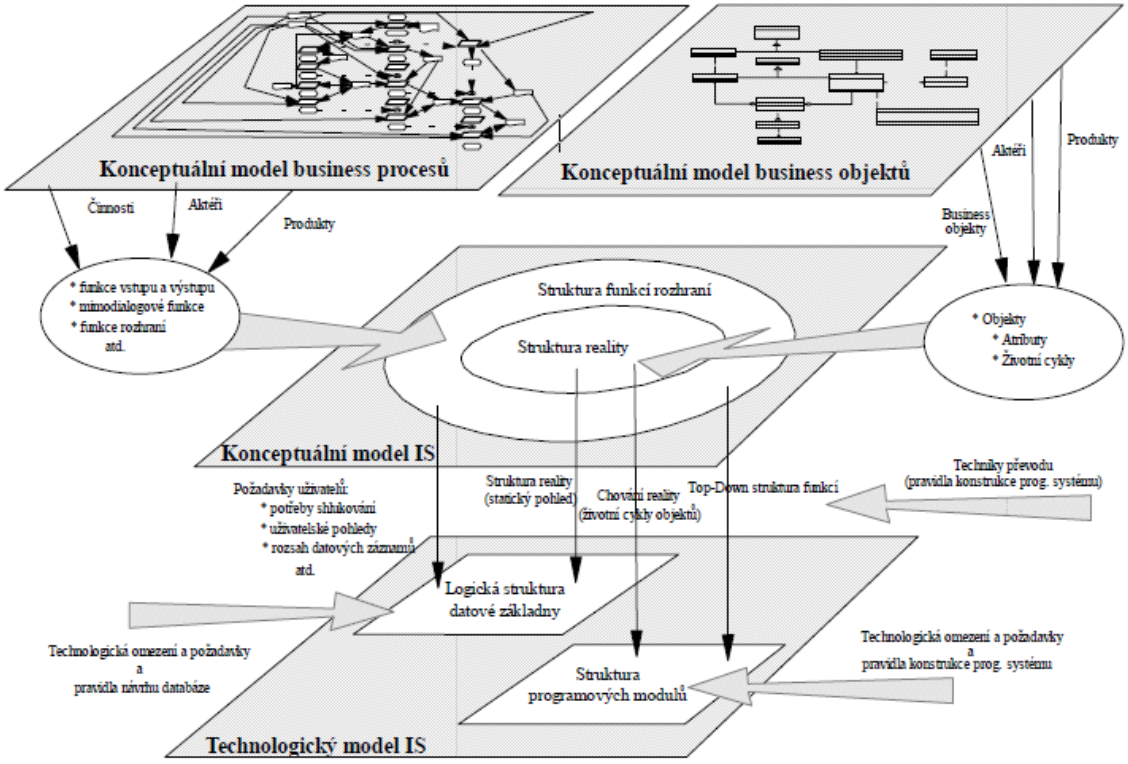
Obsah

1 Úvod

Cílem práce je seznámit se s možnostmi technologie Windows Workflow Foundation 4.0 (WF 4.0) a ukázat její možné využití v reálné aplikaci. Tato technologie byla spojena s příchodem .NET 4.0. Koncept předchozího WF 3.0, který byl uveden ve verzi .NET 3.5, byl ve verzi WF 4.0 kompletně přepracován a s původním konceptem již nemá mnoho společného. Podíváme se, co nám tato nová technologie přináší, jak nám může pomoci ve vývoji softwaru. Seznámíme se s technikami modelování podnikových procesů a krátce si představíme možnosti WF 4.0. Závěrem si ukážeme praktickou ukázku implementace vybrané části podnikového procesu pomocí této nové technologie.

1.1 WF 4.0 jako nová technologie

Většinou s novou technologií přichází i nová očekávání, jak tato technologie pomůže zlepšit stávající organizační a řídicí systémy v podniku. Mnohé podniky předstírají, že informační systémy (IS) a podnikové systémy jsou totéž, a že mnohé problémy, jež organizace mají, jsou vlastně způsobené jen nedostatečnou vybaveností tím správným IS. Ukazuje se, že ani využívání nejnovějších technologií nezaručuje úspěch. Takové zjednodušené přístupy k problematice tvorby IS a nereálná očekávání vedou k pocitu, že nový IS neslouží tak jak má. V poslední době si však mnohé organizace tento nesoulad uvědomují a snaží se vylepšit své procesy tak, aby lépe podporovaly podnikové cíle. Objevují se nové vědní disciplíny, které nabízí nejrůznější metodiky jak podnikové procesy co nejlépe zmapovat a posléze zefektivnit (BPM, Process Reingeneering atd.). Je důležité si uvědomit, že IS systém je jen jednou součástí podnikového systému a tudíž musí být navrhován v jeho celkovém kontextu (i když mohou být oblasti, kde se tyto hranice stírají). Naopak mnohdy při budování IS dochází ke změnám organizační struktury (nová oddělení) nebo ke změně způsobu řízení v organizaci. Účelem IS systému je poskytovat správné informace v reálném čase tak, aby co nejlépe podporovaly systém řízení v podniku a napomáhaly naplňování podnikových cílů. Pro popsání podnikového systému využíváme nejrůznější modely, jež nám lépe napomáhají pochopit strukturu a chování podnikového systému v organizaci.



Obr. 1.1 Analýza podnikových procesů jako východisko vývoje IS – Václav Řepa[1]

1.2 WF 4.0 a myšlenka znovupoužitelnosti programových komponent

WF 4.0 nabízí paletu nejrůznějších grafických objektů podobných těm, které využíváme při modelování podnikového systému. Znamená to tedy, že WF 4.0 je další modelovací nástroj, jenž nám pomůže lépe zachytit strukturu nebo chování podnikového systému v organizaci?

Ne tak zcela, neboť při bližším zkoumání narazíme na zdrojový kód, který se pod grafickými objekty skrývá. Představuje tedy WF 4.0 další z rodiny produktů firmy Microsoft, které dále rozvíjí myšlenku znovupoužitelnosti programových komponent?

Myšlenka znovupoužitelnosti se objevuje od 50. let minulého století. Výhody použití existujících komponent jsou zřejmé. Snižují náklady a zrychlují vývoj nových softwarových produktů. Zrychlení vývoje softwaru vede ke zkrácení času pro uvedení nových produktů na trh, což představuje významnou konkurenční výhodu. Mnohdy mnohé komponenty jsou zdarma, avšak pro

jejich efektivní použití je třeba využít placených konzultačních služeb od dodavatele komponenty.

Koncept využití již existujících programových komponent představuje střední cestu mezi vytvořením vlastního řešení, které může přinášet významnou konkurenční výhodu, avšak jeho vývoj je spojen se značnými náklady nebo zakoupením již existujícího softwaru.

Jestliže využití již existujících komponent přináší značné výhody, co brání jejich širokému uplatnění v praxi a proč se nenaplnila očekávání, která v minulosti tato myšlenka vzbuzovala?

Robert L. Glass v knize "*Loyal Opposition: Is There Really a Software Crisis?*"[2] zmiňuje, že v minulosti chybělo větší pochopení managementu, neboť nový produkt se často oceňoval dle počtu řádků zdrojového kódu a využití již existujících programových komponent představovalo pouze přidání pár řádků kódu. Později požadavky trhu kladly důraz spíše na krátké dodací lhůty, než na znovupoužitelnost dodávaných řešení, neboť vývoj znovupoužitelných komponent trval podstatně déle a byl mnohem složitější.

Clement Szypersky ve své knize „Component Software"[3] zmiňuje další úskalí při tvorbě a testování znovupoužitelných komponent. Dodavatelé komponent stojí téměř před neřešitelným problémem, jak tyto komponenty správně otestovat, neboť často ani neznají prostředí, ve kterém budou komponenty nasazeny a přidáme-li k tomu ještě rozsáhlý trh s nezávislými komponentami, pak kombinace možných konfigurací může být téměř nekonečná.

Výše uvedené překážky způsobily, že nakonec se ani nepodařilo vytvořit dostatečný trh se znovupoužitelnými komponentami, který by dokázal uspokojit rozmanité potřeby trhu. Přesto byly společnosti, které dokázaly úspěšně využít tuto myšlenku a např. firma „Software engineering laboratory" v NASA-Goddard dokázala využít až 80% svých předchozích řešení. Příkladem dalšího úspěšného produktu byl jistě Visual Basic od firmy Microsoft, který zejména v 90-tých letech byl velice populární.

2 Seznámení se s workflow

Hlavním smyslem workflow systému je automatizace a počítačová podpora všech možných procesů v organizaci. Je důležité si uvědomit, že efektivní nasazení informačního systému je podmíněno právě korektním návrhem procesu. Podpora modelování a řízení procesů je dalším smyslem workflow systému. Organizace může obsahovat nejeden informační systém a workflow může sloužit k integraci informačního prostření v celém podniku.

Ukážeme si typy byznys procesů v organizaci a typy workflow systémů. Na těchto ukázkách uvidíme, jak jsou pojmy byznys proces a workflow provázané a proč jsou často považovány za synonyma.

2.1 Pojmy

Na začátek si definujeme základní pojmy, které budeme používat.

Byznys proces (podnikový proces) je uspořádaná množina aktivit a procedur. Tato množina je složena za účelem realizace podnikatelského záměru. Podnikatelský záměrem může být stavba budovy, pečení chleba nebo i trošku abstraktnější činnost např. řešení pojistné události nebo soudní spor. Pojem *procedura* chápeme jako podproces v jiném procesu. *Aktivita* je popis činnosti ve vykonávaném procesu, je atomická (dále nedělitelná) a reprezentuje jeden krok v procesu. Díky proceduře můžeme zachytit byznys proces na různé abstraktní úrovni. Aktivity a procedury se nemusí řadit do jedné posloupnosti, ale můžou být i v několika posloupnostech vedle sebe. To umožňuje zpracovávat procesy paralelně.

Pro definici *procesu* můžeme použít neformální nebo formální cestu. Při použití neformální cesty, použijeme přirozený jazyk, obrázky, tabulky a jiné nástroje pro pochopení procesu. Pokud použijeme formální cestu, tak si musíme vybrat techniku, kterou proces zachytíme jednoznačně. Tyto techniky mají precizně definovanou syntaxi a sémantiku, které umožňují nad nimi provádět různé typy verifikace. Takto definované procesy se hodí pro programové zpracování a tím automatizování těchto procesů.

Při vytváření *modelu byznys procesu* záleží na míře abstrakce, kterou použijeme. V určitém procesu může existovat aktivita, která se na nižší míře abstrakce přemění na proceduru. Tato hierarchizace umožňuje přehledně

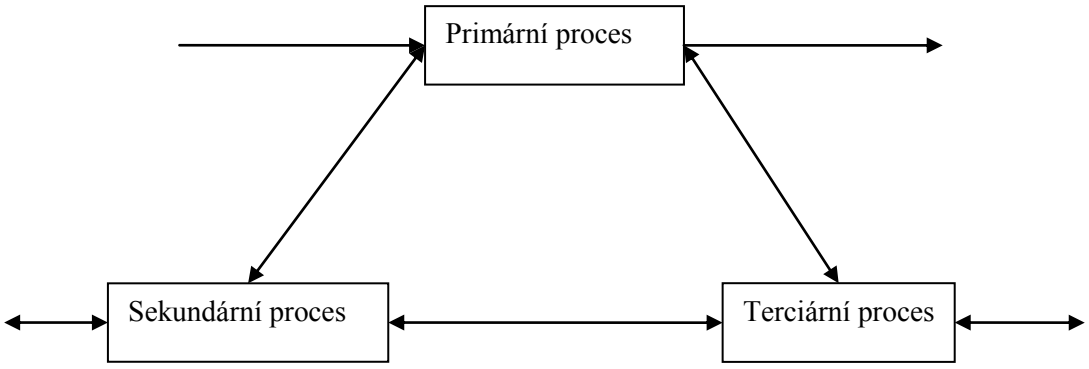
zachytit proces. Při modelování bychom měli vytvořit takový náhled na proces, který umožňuje pochopit všechny aktivity a souvislosti mezi nimi.

Instance procesu je chápána jako konkrétní případ vykonání procesu. Proces má přesně definovaný začátek a konec. Existují procesy, které se opakují ve smyčce dokola, říká se jim kontinuální procesy. Ale i u těchto procesů se dá vysledovat počátek a konec periody. Po ukončení procesu je výstupem konkrétní produkt. Proces můžeme brát, jako předpis pro instanci procesu, podle kterého vznikne produkt. Snažíme se navrhnout proces tak, že každá instance stejného procesu vytvoří stejný produkt, což nám zaručuje stabilitu a kvalitu produktu.

Workflow je téměř to samé, jako byznys proces. Jediný rozdíl mezi nimi je v tom, že workflow je automatizovaný. Tím, že je to automatizovaný byznys proces, tak se dá spravovat a řídit programově. Z důvodů toho, že workflow se dá zpracovat programy, jsou při jeho definování kladené vysoké nároky na přesnost a jednoznačnost. Programy zabývající se správou a řízením byznys procesů se nazývají ERP (Enterprise Resource Planning) a WFM (Workflow Management).

2.2 Typy byznys procesů

Aalst, W., Hee, K. ve své knize „*Workflow Management: Models, Methods and Systems*.“ vysvětluje typy byznys procesů.



Obr. 2.1 Vazby mezi třemi typy procesů – Aalst, W., Hee, K.[4]

- Primární procesy, nebo také výrobní procesy, jsou ty, které produkují výrobky společnosti. Tyto výrobní procesy jsou jasně orientované na zákazníka a jsou tím pádem přímým finančním zdrojem společnosti. Někdy není zákazník dosud znám a výrobky jsou produkovány do zásoby. Primární procesy můžou být např. projekční, distribuční, výroba, skladování, prodej, a jiné.

- Sekundární procesy neboli podpůrné procesy jsou zaměřeny na podporu primárních procesů. Hlavním cílem podpůrných procesů je, aby nepoklesl výkon primárních procesů z důvodů nedostatku nějakých zdrojů. Podpůrné procesy můžeme rozdělit na čtyři části: nákup a údržba strojů, lidské zdroje, marketing a finanční správa.

- Terciární procesy nebo také řídící procesy slouží pro řízení primárních a sekundárních procesů. Z primárních a sekundárních procesů získávají data, která vyhodnocují a poté vytyčují cíle pro dané procesy. Řídící procesy také přiřazují procesům zdroje a udržují kontakty se zainteresovanými stranami.

Obr. 2.1 znázorňuje vztahy mezi těmito třemi typy procesů. Vstupem řídících procesů je kapitál a podnikatelský cíl. Výstupem těchto procesů je zajištění zisku. Podpůrné procesy dostávají od řídících procesů prostředky k nákupu zdrojů, ty potom mohou poskytovat výrobním procesům. Některé zdroje poskytnuty výrobním procesům se po použití vracejí. Vstupem výrobních procesů jsou objednávky a suroviny, ze kterých vzniká výrobek nebo také služba. Výrobní a podpůrné procesy poskytují zpětnou vazbu řídícím procesům v podobě dat, která jsou potom zpracována a vyhodnocena. Výrobní procesy obvykle mají diskrétní charakter. Obdrží objednávku, kterou realizují a poté předají. Podpůrné a řídící procesy jsou často spojité, i když v sobě mohou obsahovat dílčí diskrétní procesy.

2.3 Typy workflow

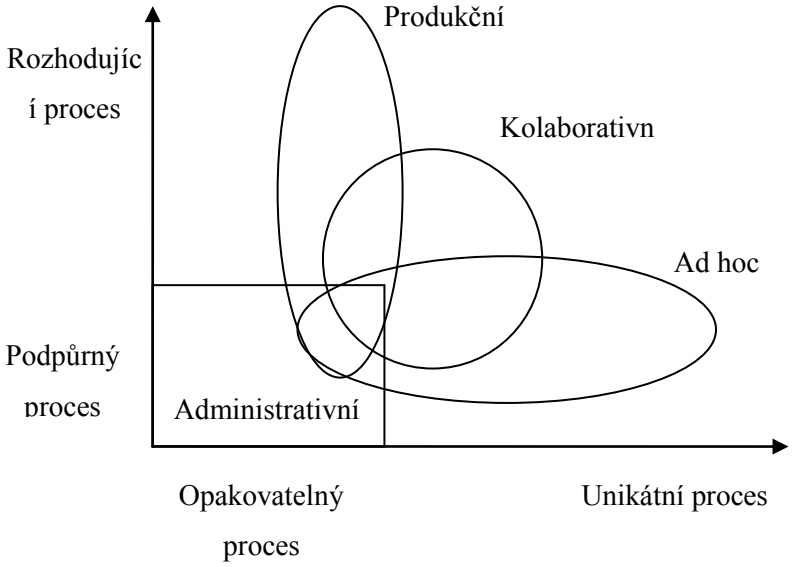
Martin Mayer, v první části své diplomové práce *„Informační podpora procesů krizového řízení“*[5], se zabývá rozdělením workflow na čtyři různé základní typy dle charakteru procesů.

- Produkční workflow podporují především primární procesy v podniku. Procesy bývají velmi dobře formalizované a přesně strukturované. Tyto procesy probíhají v podniku velmi často, a proto jsou většinou velmi dobře optimalizované. Důraz je kladen na vysokou produktivitu.

- Administrativní workflow slouží pro sekundární procesy v podniku. Slouží nejčastěji k vyřizování každodenních administrativních činností. Procesy jsou jednoduché, podrobně strukturované a mají málo alternativních možností. Tyto procesy jsou většinou spjaté s formuláři nebo s jinými dokumenty. Účastníci jsou příležitostní a v každé organizaci můžou být odlišné činnosti. Nutná je dostupnost pro všechny pracovníky organizace, protože je využívá většina zaměstnanců.

- Kolaborativní workflow je zaměřeno především k řízení spolupráce účastníků v procesu. Procesy jsou méně strukturované a prolíná je kreativita účastníků. Charakterizuje je dynamická změna definice procesů.

- Ad hoc workflow představuje procesy, jejichž průběh není předem definovaný. Tento typ procesů je většinou unikátní a definován samotnými účastníky, proto musí být snadno definovatelné. Příkladem ad hoc procesu je například úprava zboží přímo pro konkrétního zákazníka.



Obr. 2.2 Typy workflow – Mayer[5]

Při nasazení workflow systému je nutné vybrat správný typ produktu pro konkrétní řešení. Pro výkonnost vybraného workflow je rozhodující výběr správného produktu. Při tomto výběru nás možná napadne myšlenka, že by mohlo existovat jedno univerzální řešení. Tady ovšem narazíme na rozdílnost workflow typů. Ad hoc a kolaborativní řešení se vyznačují jednoduchou strukturou a vysokou mírou možnosti zásahu účastníka. Opakem jsou produkční a administrativní řešení. Ta se vyznačují složitou mírou struktury a tím pádem velmi malou možností zásahu účastníků do procesu. Výrobci workflow produktů se snažili o sjednocení všech typů. Snažili se do produkčního systému vložit dynamické změny a možnost změnit organizační strukturu, tady se jim to částečně povedlo. Ale do kolaborativních produktů zaváděli složitější struktury, což vedlo k zavedení složitých a neefektivních procesů. Vize vytvoření univerzálního produktu, který by byl dynamický s vysokou mírou spoluúčasti a dostatečně strukturovaný, se prozatím nepovedla.

2.4 Modelování procesů

Nezbytnou součástí workflow je modelování. V modelování se používá několik základních technik, které si ukážeme na jednoduchých příkladech. Existuje mnoho nástrojů k modelování procesů a my použijeme Petriho sítě, které si v krátkosti představíme.

2.4.1 Základní techniky při modelování byznys procesů

Základní stavební jednotkou při modelování byznys procesů je aktivita. Můžeme ji rozdělit na tři typy:

- Aktivita, kterou vykonává pouze člověk (neautomatizovaná)
- Aktivita, kterou vykonává člověk ve spolupráci se stroji (polo automatizovaná)
- Aktivita, kterou vykonává jen stroj (plně automatizovaná)

Při vytváření byznys modelu se používají tyto základní techniky:

- Sekvence – je sériové uspořádání aktivit jdoucí přímo za sebou. Sekvence je asi nejpoužívanější konstrukce při modelování byznys procesů.
- Iterace - je opakování aktivity v cyklu.
- Alternativa - slouží k výběru jedné aktivity z více nabízených. Alternativu si můžeme také představit jako logické „nebo“.
- Paralelismus - umožňuje provádění více aktivit souběžně. Můžeme si jej představit jako logické “a zároveň“.

Vzájemná návaznost aktivit se modeluje pomocí šipek. Šipka je v podstatě orientovaná hrana, která vede od aktivity k aktivitě.

Při modelování postupujeme od nejvyšší úrovně k nejnižší. Na nejvyšší úrovni abstrakce modelem zachycujeme jen nejdůležitější činnosti popisovaného procesu a aktivity jsou většinou chápány jako procedury, které se dále zpřesňují. Na nejnižší úrovni se vyskytují nedělitelné aktivity a pomocí nich jsou modelovány detaily samotného procesu.

2.4.2 Modelování byznys procesů pomocí Petriho sítí

Existuje mnoho nástrojů pro modelování byznys procesů. Všechny tyto nástroje si můžeme rozdělit dle specifikace modelovaného systému, a to na formální a neformální. Neformální metoda používá pro zachycení procesu přirozený jazyk, tabulky, obrázky a jiné techniky. Některé neformální metody mají přesně danou syntaxi, ale jejich sémantika není úplná. Těmto metodám se taky někdy říká semiformální. Teď už je asi jasné, jak jsou definované formální metody, mají přesně definovanou syntaxi i sémantiku. Co to pro nás znamená?

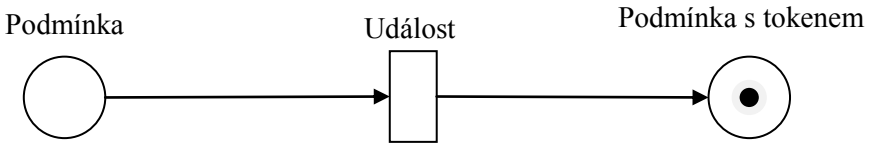
Definovaná syntaxe nám říká, jakými prvky můžeme modelovat byznys proces a sémantika, jak je máme chápat. Pokud má metoda přesně definovanou sémantiku, tak vytvořený model chápou všichni stejně a nemůže si ho vykládat každý jinak. U takto vytvořeného modelu můžeme provádět různé verifikace. Formální metody můžeme rozdělit na dva styly, první je funkční a druhý je popisný. Jelikož vytváříme modely, tak se budeme zajímat o funkční styl. Nejznámější nástroj pro funkční popis systému je konečný automat. Používání formálních metod pro vytvoření modelu vyžaduje veliké úsilí a čas. S tím jsou spjaté veliké finanční náklady, a proto se využívají pro systémy, u kterých je nezbytný bezchybný chod programu. U velkých projektů by vytvoření celého modelu bylo velmi nákladné, proto můžeme tímto způsobem namodelovat jen nejdůležitější části.

„Jedním z formálních nástrojů pro byznys modelování je Petriho síť. Petriho sítě vznikly za účelem rozšíření modelovacích možností konečných automatů. Jejich autorem je prof. Carl Adam Petri, který je navrhl jako nástroj určený k modelování a analýze procesů“ [6].

V průběhu času vznikalo mnoho rozšíření Petriho sítí, tak aby jeho modelovací schopnost vyhověla praktickým případům, a v dnešní době je to komplexní a silný modelovací nástroj. My se omezíme na nezbytné představení Petriho sítě pro účely modelování byznys procesů. S rozšiřováním Petriho sítí, začaly vznikat přívlastky pro každé rozšíření sítě. Petriho sítím zaměřeným na modelování byznys procesů, se říká C/E (Condition/Event) Petriho síť. Doc. Markl se ve svých sylabech *„Petriho sitě 1“* [7] zabývá tímto modelovacím nástrojem.

„C/E Petriho síť je zadána následujícími údaji (a na základě těchto údajů může být také graficky zobrazena):

- *Podmínkami (conditions) zobrazovanými kroužky*
- *Událostmi (events) zobrazovanými obdélníky (případě úsečkami)*
- *Šipkami vedoucími od podmínek k událostem*
- *Šipkami vedoucími od události k podmínkám*
- *Tokený (zobrazenými tečkami v kroužcích podmínek) indikujícími logický stav (pravdivost) podmínek. Jejich počáteční rozložení v síti nazýváme počátečním stavem nebo počátečním značením (initial marking) sítě "* [7]



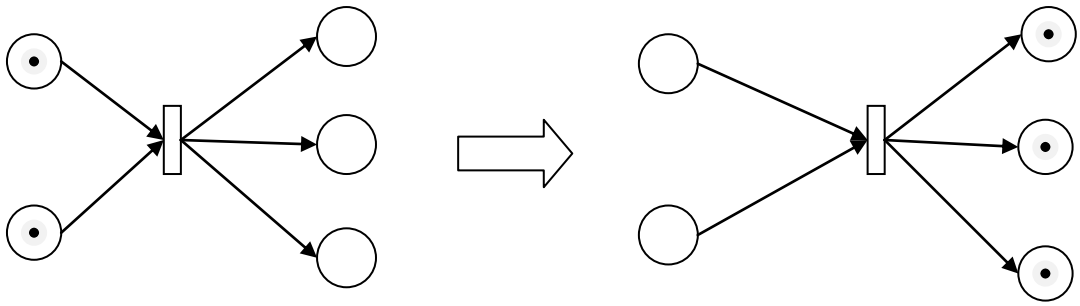
Obr. 2.3 Grafické zobrazení prvků Petriho sítě

„V C/E Petriho síti:

- *Podmínka c je vstupní podmínkou (precondition) události e, jestliže od podmínky c vede šipka k události e*
- *Podmínka c je výstupní podmínkou (postcondition) události e, jestliže od události e vede šipka podmínce c*
- *Každá podmínka je vždy buď splněna, nebo nesplněna*
- *Každá splněná podmínka je indikována tokenem, tj. tečkou uvnitř kroužku zobrazujícího podmínku*
- *Celkový (globální) stav sítě je zadán množinou podmínek, které jsou v daném okamžiku splněny*

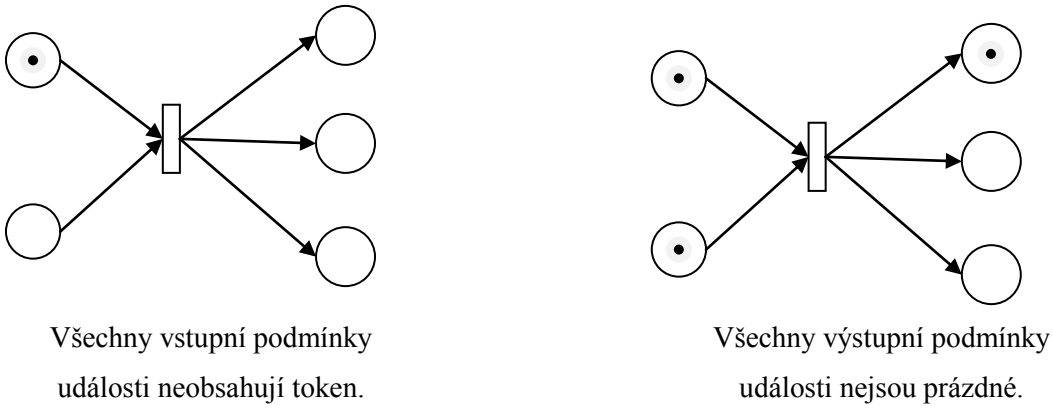
Změny stavu C/E Petriho sítě probíhají podle následujících pravidel:

- *Ke změnám stavu sítě dochází uskutečňováním událostí*
- *Enabling rule: událost může nastat, jsou-li všechny její vstupní podmínky splněny a současně všechny její výstupní podmínky nesplněny – takovou událost nazýváváme proveditelnou*
- *Provedena může být pouze proveditelná událost, proveditelná událost může také zůstat neprovedena*
- *Firing rule: po provedení proveditelné události jsou všechny její výstupní podmínky splněny a všechny její vstupní podmínky nesplněny"* [7]



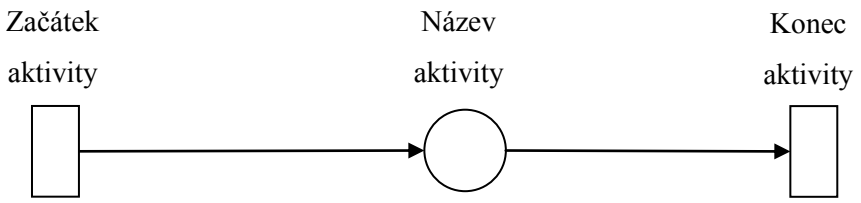
Obr. 2.4 Změna stavu po provedení proveditelné události - Markl [7]

Pravidla pro provedení události jsou jasně daná. Při provedení události se ze vstupních podmínek zkonzumují tokeny a na výstupních podmínkách se vyprodukují. Pokud alespoň jedna vstupní podmínka nemá token, tak se událost nemůžeme provést. Ve výstupních podmínkách se nesmí nacházet žádný token, jinak se událost neprovede, protože nemá kam vložit vyprodukovaný token. Z toho nám vyplývá, že C/E Petriho síť neumožňuje mít více tokenů v jedné podmínce.

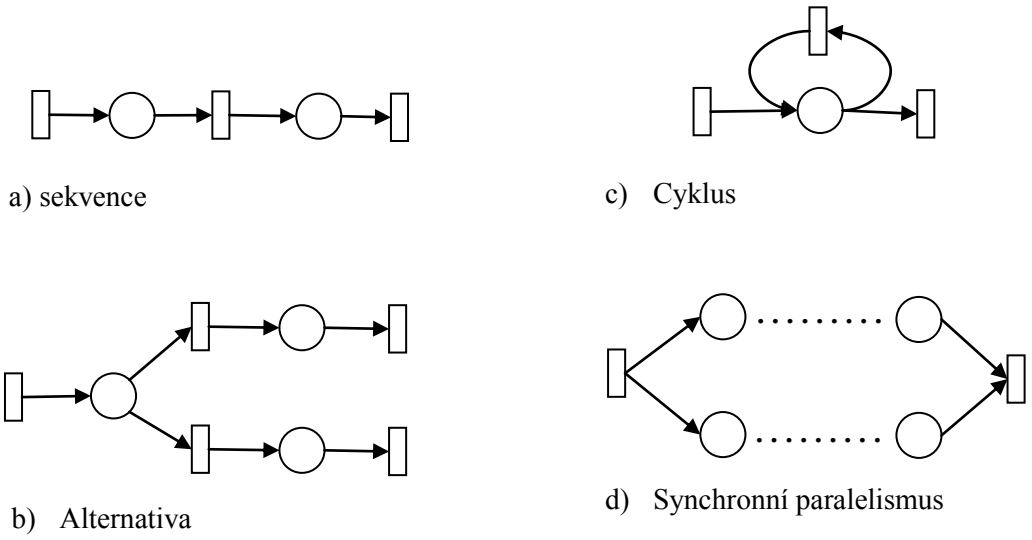


Obr. 2.5 Zobrazení neproveditelných událostí

V předchozí části jsme si definovali základní prvky a techniky, které slouží k vytvoření byznys procesů. Každý modelovací nástroj realizuje tyto abstraktní konstrukce jinak. Nyní se podíváme, jak jsou realizované pomocí C/E Petriho sítě. Základním stavebním prvkem je aktivita, která má začátek a konec aktivity reprezentovaný událostí a mezi nimi je podmínka viz obr 2.6. Na obrázku 2.7 jsou znázorněné základní techniky. Sekvence je asi nejpoužívanější technikou a jedná se o řazení aktivit za sebou, kde konec první aktivity je zároveň začátkem druhé. Alternativa je realizovaná podmínkou, která má více možných výstupních událostí. Na tyto události se můžou vázat další aktivity. Cyklus umožňuje provádět aktivitu vícekrát za sebou, dle potřeby. Synchronní paralelismus slouží pro modelování dvou synchronních procesů. Vzniká tak, že událost má více výstupních podmínek a tím proces rozvětví. Podmínkou ovšem je, že po provedení těchto procesů se synchronizují pomocí koncové události a dále už pokračuje jen v jedné větvi. V Petriho sítích se v jednom kroku může provést jen jeden „tah“. Pokud se nám tedy proces rozvětví, tak se události neprovádí souběžně obě naráz, ale nejdřív se provede jen jedna a poté druhá. Rozhodnutí, která to zrovna bude, je čistě náhodná věc. Ta samá náhoda rozhoduje o výběru i v alternativě. Náhodně se zvolí aktivita, která se provede a tento společný token zkonzumuje.

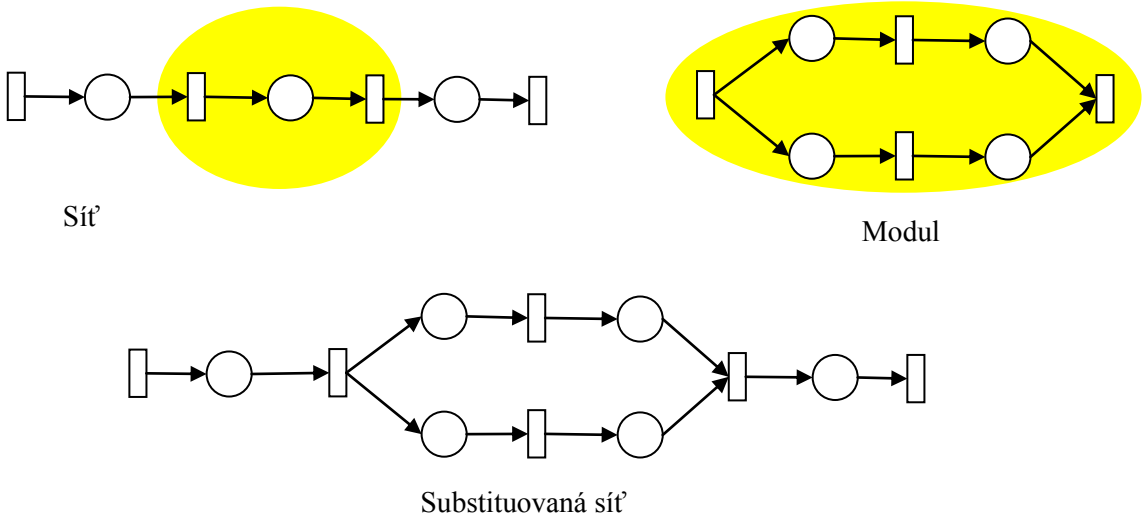


Obr. 2.6 Aktivita



Obr. 2.7 Základní prvky modelování

Důležitou součástí modelování je i zachycení různé míry abstrakce byznys modelu. K tomu slouží hierarchizace, kde na jedné úrovni abstrakce máme aktivitu, která se ovšem na nižší úrovni stává procedurou. Proceduře říkáme modul, který nám reprezentuje další Petriho síť. Existují ovšem nějaké zásady, jak substituovat moduly za aktivity. Aktivita začíná a končí událostí. Pokud budeme chtít modul substituovat za aktivitu, tak modul musí začínat a končit taktéž událostmi. Pokud bychom ovšem měli modul, který takto nezačíná nebo nekončí, přidáme si potřebné komponenty, abychom toho dosáhli. Tímto postupem můžeme vytvářet různé úrovně abstrakce a docílit tak lepší přehlednosti modelu.



Obr. 2.8 Zapojení modulu do sítě

3 Základní možnosti WF 4.0.

V předchozí kapitole jsme se zabývali wokflow a jeho rozdělením na čtyři základní typy. Pokud bychom WF 4.0 chtěli zařadit do jednoho z nich, tak patří mezi produkční workflow. Je přímo zaměřené na výrobu, přesněji na výrobu softwarového produktu.

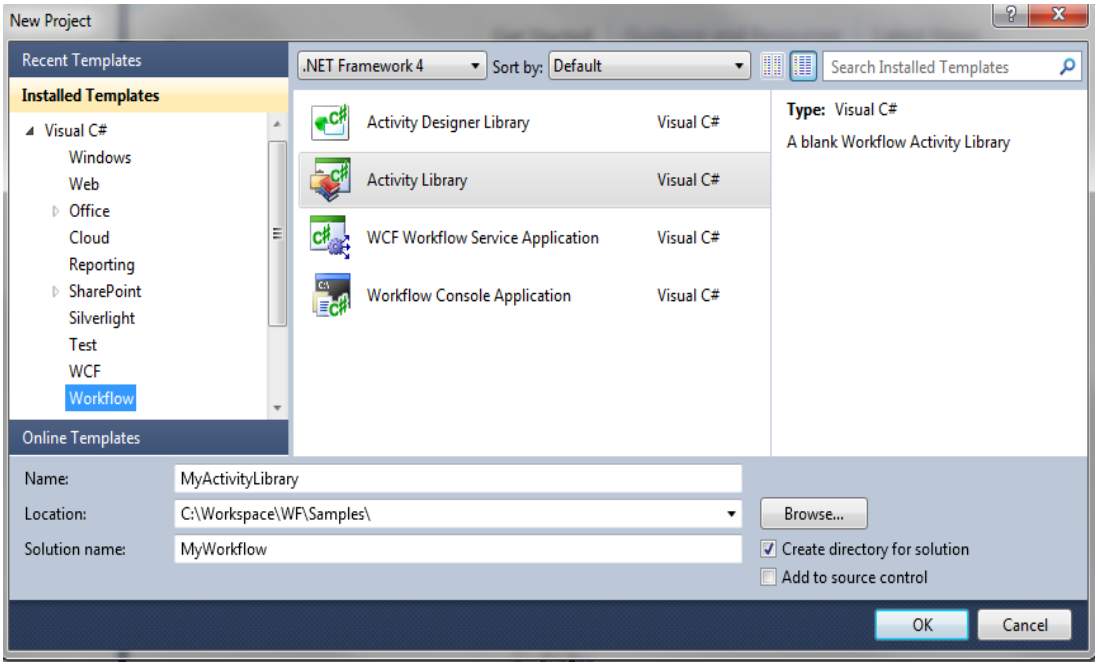
WF 4.0 má vytvořené aktivity, které můžeme využít při implementaci procesu, a tím nám usnadní práci. Skládáním aktivit si můžeme vytvořit vlastní aktivitu. V této kapitole si ukážeme, jaké aktivity už jsou předvytvořené a k čemu se nám hodí. Ukážeme si to na jednotlivých příkladech složených aktivit. Před tím si ještě vytvoříme knihovnu, kde budeme mít vlastní aktivity a aplikaci, ve které budeme ukázky spouštět.

3.1 Vývojové prostředí pro práci s WF 4.0

Microsoft Visual Studio (VS) je univerzální vývojové prostředí pro platformu .NET. Ve verzi VS 2010 je jeho součástí i podpora pro práci s WF4.0. Obsahuje předpřipravené šablony pro nejčastější využití WF4.0 a designér, pomocí kterého můžeme vytvářet vlastní aktivity nebo celá ucelená řešení. V této části si na příkladech předvedeme použití základních šablon, které posléze využijeme v dalších ukázkách. Obeznámíme se s designérem a jeho možnostmi.

Postup vytvoření knihovny s aktivitami

1. Spustíme si VS 2010 a vybereme *New project*
2. Vybereme si z *Installed Template* záložku *Wokflow* a zobrazí se nám nabídka čtyř možných předloh
3. Vybereme předlohu *Activity Library* a pojmenujeme si ji *MyActivityLibrary***.** Solution name si nazveme MyWorkflow



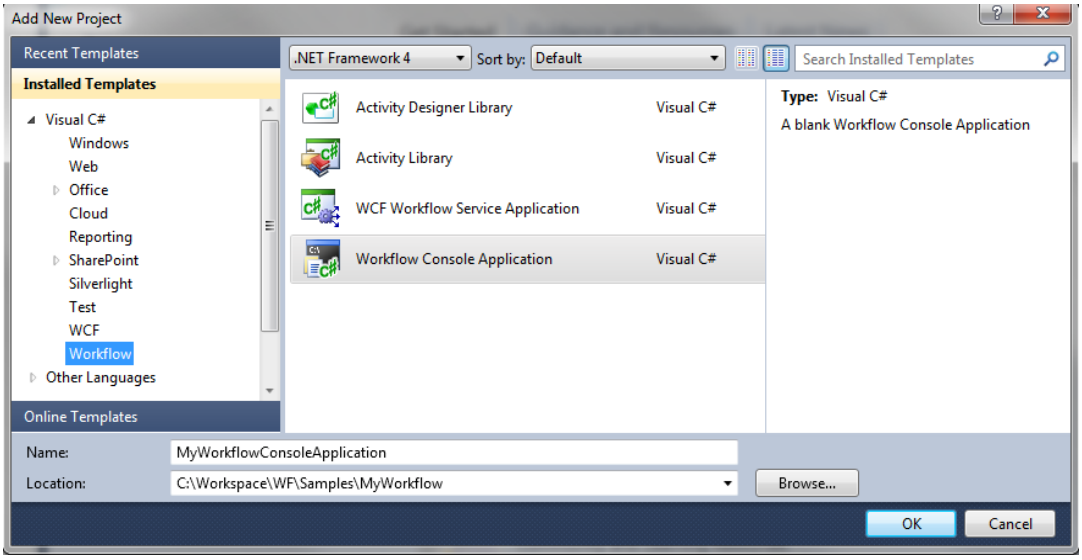
Obr. 3.1 Vytvoření knihovny

4. Odstraníme předpřipravenou *Activity1.xaml*. Abychom měli čistou knihovnu, do které budeme vkládat naše vlastní aktivity

Vytvořili jsme si knihovnu, do které budeme vkládat všechny naše aktivity. Tuto knihovnu potom můžeme využívat i v jiných projektech.

Vytvoření workflow konsolové aplikace s naší knihovnou aktivit

1. Myší najedeme na MyWorkflow v solution exploreru a stiskneme pravé tlačítko myši. V kontextovém menu vybereme add, a poté new project
2. Vybereme si z *Installed Template* záložku *Wokflow* a zobrazí se nám nabídka čtyř možných předloh
3. Vybereme předlohu *Workflow Console application* a pojmenujeme si ji *MyWorkflowConsoleApplication*



Obr. 3.2 Vytvoření workflow konsolové aplikace

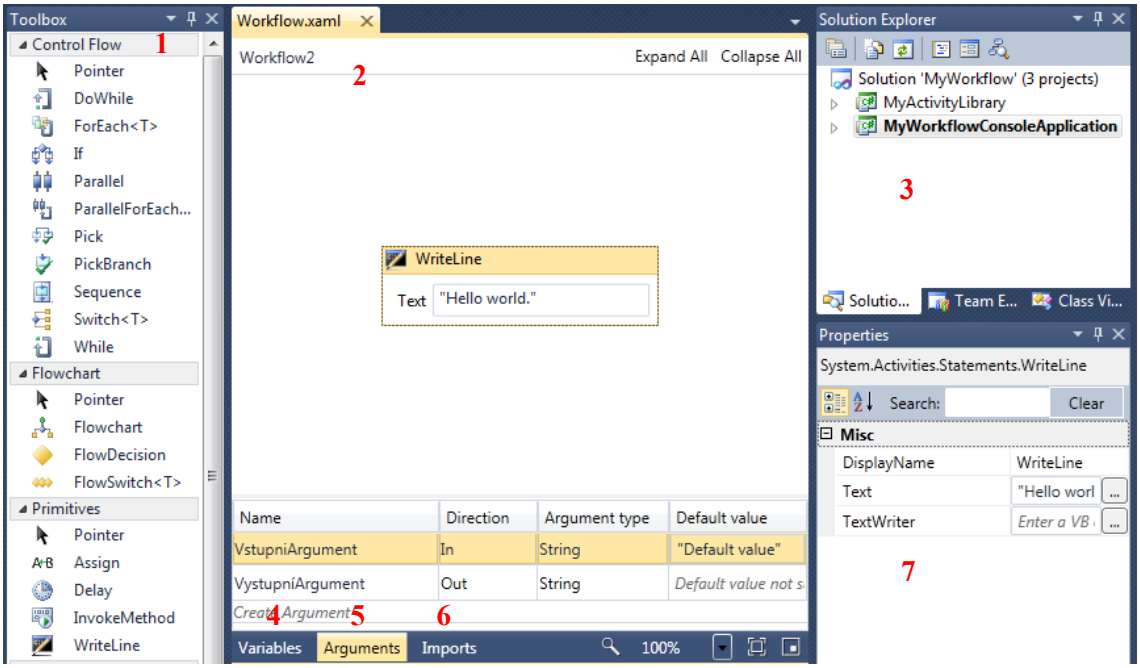
4. Dvojklikem otevřeme Program.cs a na konec metody *Main* přidáme Console.ReadKey();
5. Stiskneme F6 a zkompilujeme všechny projekty

```
using System;
using System.Linq;
using System.Activities;
using System.Activities.Statements;

namespace MyWorkflowConsoleApplication
{
class Program
      {
static void Main(string[] args)
            {
            // spuštění workflow
            WorkflowInvoker.Invoker(new Workflow1());

            // slouží pro zastavení výpisu na konsolu
            Console.ReadKey();
            }
      }
}
```

Kód 3.1 Ukázka spuštění Workflow1 v Program.cs

Vytvořili jsme konsolovou aplikaci, která bude spouštět naše workflow. Spouštění workflow probíhá v Program.cs, kde jsme taky přidali řádek pro zastavení výpisu kódu na konsolu. Tuto aplikaci budeme používat pro spuštění našich aktivit, abychom mohli vidět, co naše aktivity dělají.

Visual Studio 2010



Obr. 3.3 Designér ve VS 2010

1- Toolbox – obsahuje komponenty určené k využití tvorby aplikace. Můžeme si vybrat dva typy modelování: pomocí komponent Control Flow nebo Flowchart
2- Hlavní okno – slouží k vytváření modelu
3- Solution Explorer – umožňuje náhled na obsah projektu
4- Variables – zde se vytváří lokální proměnné
5- Argument – využívá se k vytvoření vstupních a výstupních argumentů workflow
6- Import – slouží k importování externích knihoven
7- Properties – zobrazuje vstupní a výstupní argumenty a jiné hodnoty aktivty

3.2 Základní modelovací techniky realizované ve WF 4.0

Pomocí Petriho sítí jsme si přiblížili základní prvky používané k modelování. Nyní si jejich realizaci představíme ve WF4.0. Vytvoříme si několik jednoduchých příkladů, na kterých si předvedeme, jak se s WF4.0 pracuje a co umí.

3.2.1 Aktivita

Vlastní aktivity ve WF4 si můžeme vytvořit pomocí programového kódu. K tomu nám slouží knihovna System.Activies. Knihovna nabízí několik abstraktních tříd, které využijeme k vytvoření vlastní aktivity.

Zejména jde o tyto třídy:

- CodeActivity – nejpoužívanější, protože pomocí ni realizujeme většinu aktivit.
- NativeActivity – obsahuje oproti CodeActivity funkci bookmark. Bookmark umožňuje uložit si stav aktivity a posléze se k němu vrátit. Toho se využívá u aktivit, které komunikují s periferiemi a čekají na odpověď. Klasickým příkladem je komunikace se serverem nebo čekání na odpověď od uživatele.
- AsyncCodeActivity – využívá se u aktivit, které se mají provádět na pozadí aplikace.

Jako ukázku si vytvoříme vlastní aktivitu, která bude vypisovat zadaný text na konzulu. K její realizaci využijeme třídu CodeActivity.

Postup vytvoření vlastní aktivity:

1. Najedeme myší na projekt *MyActivityLibrary* a stiskneme pravé tlačítko myši. Rozbalí se nám kontextové menu a v něm vybereme *add* a *new item.*
2. Vybereme z *Installed Template* záložku *Wokflow*
3. Vybereme předlohu Code Aktivity a pojmenujeme si ji MyCodeAktivity.cs**.**
4. Do metody Execute přidáme Console.WriteLine(text);
5. Přeložíme aktivitu.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Systém.Activites;

namespace MyActivityLibrary

{

      Public sealed class MyCodeActivity: CodeActivity
      {
            // vstupní argument aktivity
            Public InArgument<string> Text {get; set; }

            Protected override void Execute (CodeActivityContext
      context)
            {
            //Získání hodnoty vstupního argumentu z contextu
            string text = context.GetValue(this.Text);
            Console.WriteLine(text);
            }

      }

}
```

Kód 3.2 Ukázka Code aktivity

V předloze Code Aktivity máme vytvořený vstupní argument *Text* a metodu s názvem *Execute*, která se zavolá při spuštění aktivity. Pokud bychom chtěli vytvořit i výstupní argument, tak to bude stejné jako u vstupního, jen s tím rozdílem, že místo *InArgument* použijeme *OutArgument* a pro nastavení se použije motoda *SetValue*(*OutArgument*) obsažena v context. Nyní máme v naší knihovně aktivit vytvořenou vlastní aktivitu.

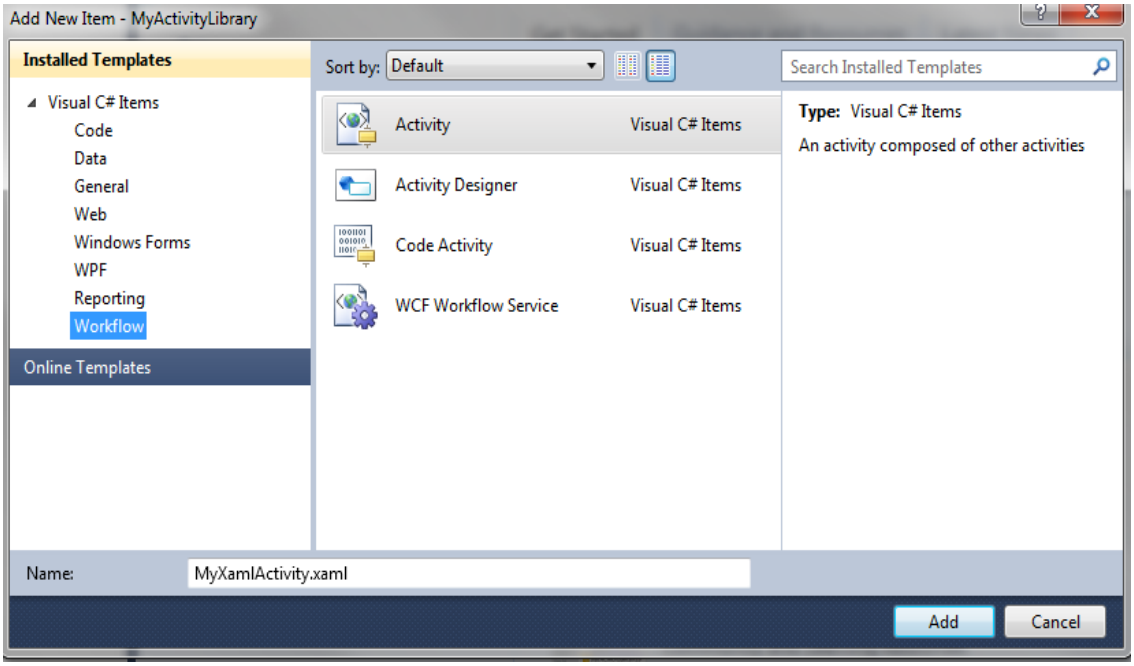
Aktivity nemusíme vytvářet jen pomocí programového kódu, ale můžeme je vytvářet i skládáním již vytvořených aktivit. WF4 nabízí ke skládání aktivit designer, ve kterém jsou dva módy, které jsou vzájemně kompatibilní:

- Control flow
- Flowchart

Oba módy mají předpřipravené aktivity, které můžeme využívat při vytváření svých vlastních aktivit. Nyní si ukážeme jak vytvářet aktivitu v designéru.

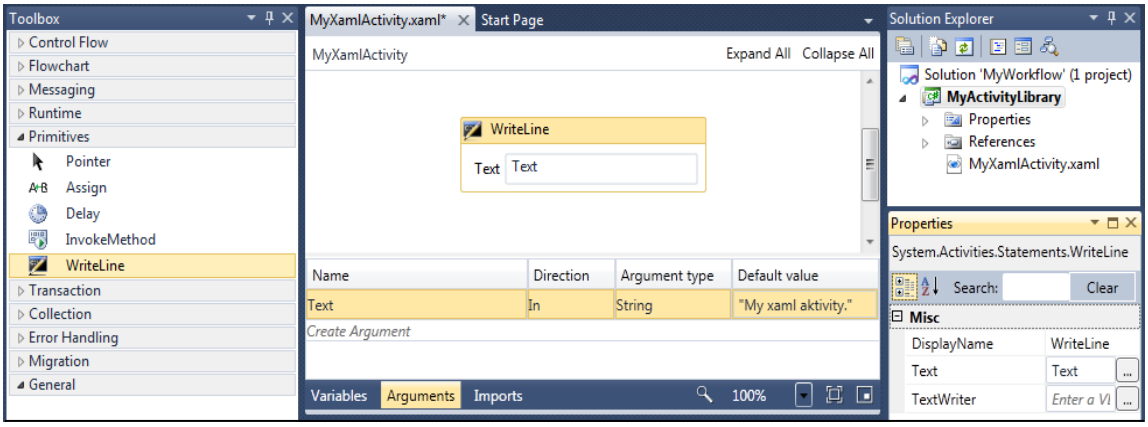
Postup vytvoření xaml aktivity:

1. Najedeme myší na projekt *MyActivityLibrary* a stiskneme pravé tlačítko myši. Rozbalí se nám kontextové menu a v něm vybereme *add* a *new item.*
2. Opět vybereme z *Installed Template* záložku *Wokflow.*
3. Vybereme předlohu *Activity* a pojmenujeme si ji *MyXamlActivity*.



Obr. 3.4 Nabídka šablon pro vytváření workflow

4. V *Toolboxu* v sekci *Primitives* přetáhneme do hlavního okna komponentu *Writeline*.
5. V hlavním okně na spodní liště klikneme na *Aruguments* a otevře se nám seznam s vytvořenými argumenty. Vytvoříme argument s názvem *Text*, směr bude *in*, typ bude *string* a *default value* bude "My xaml aktivity".



Obr. 3.5 Vytvoření aktivity

6. Stiskneme tlačítko F6, ať se nám aktivita přeloží.

Nyní jsme vytvořili jednoduchou xaml aktivitu, která nám vypíše text na konzolu. Má vstupní argument s názvem *Text*, přes který můžeme zadat aktivitě text, který chceme nechat vytisknout na konzolu. Stejným způsobem můžeme vytvořit i výstupní argument, jen místo *in* v direction vybereme *out*. To se nám může hodit, pokud aktivita poskytuje nějaké výstupní data.

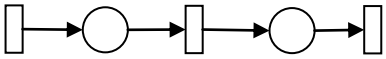
Můžeme se podívat, co nám designer vygeneroval za kód na pozadí aplikace. Najedeme myší v solution exploreru na MyXamlAktivity.xaml a stiskneme pravé tlačítko myši. Zobrazí se nám kontextové menu a vybereme view designer. Zobrazí se nám xaml kód v hlavním okně.

Xaml kód je založen na xml. Microsoft ho používá od .NET Frameworku 3.0 a to v technologiích WPF a silverlight pro popis GUI. Ve WF využívají xaml k popisu samotného workflow. XAML můžeme zkompilovat do .baml souboru (Binary XAML), který potom můžeme používat v .NET projektu jako resource.

3.2.2 Sekvence

Sekvence je nejpoužívanější technika při modelování. K realizaci v Control Flow je k tomu připravena komponenta sequence. Do sequence můžeme sériově vkládat aktivity. Ve FlowChart se sekvence realizuje vložením aktivit a propojením šipkami určíme pořadí vykonávání aktivit.

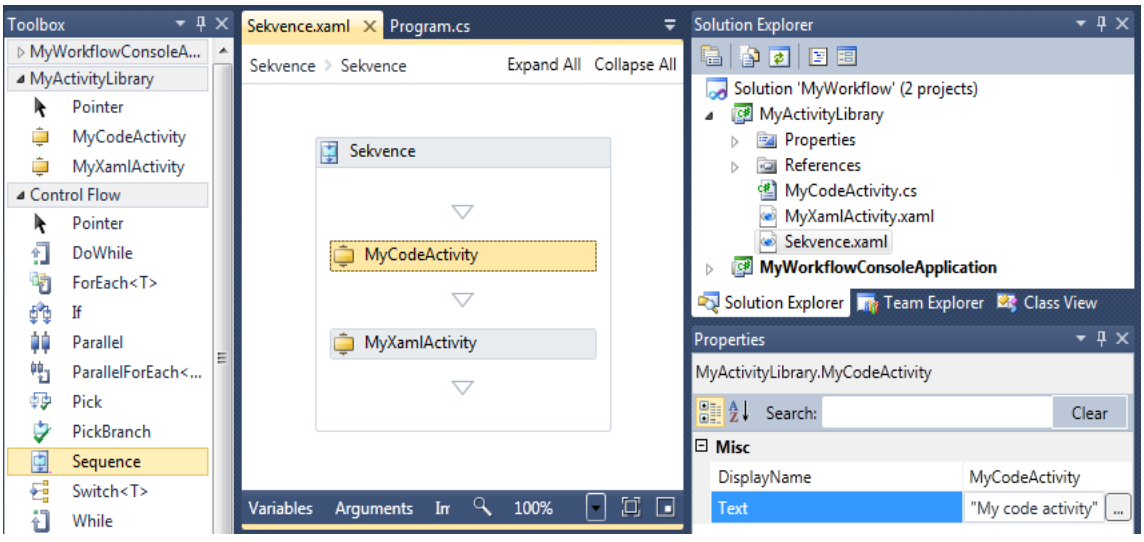
Control flow sekvence



Obr. 3.6 Příklad obsahující sekvencí

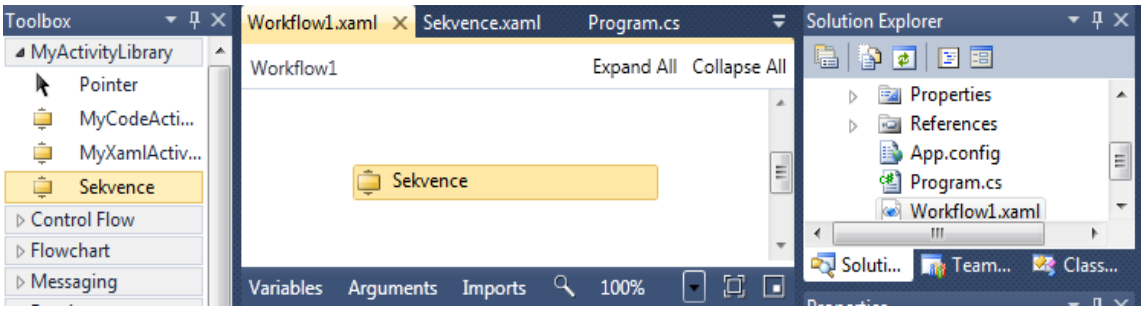
Postup realizace:

1. Najedeme myší na MyActivityLibrary a známým způsobem přidáme xaml aktivitu s názvem Sekvence. Po přidání se nám otevře hlavní okno přidané aktivity.
2. Z toolboxu v sekci Control Flow vezmeme komponentu Sequence a přetáhneme ji do hlavního okna. Přejmenujeme si ji na Sekvence. Můžeme to udělat dvojklikem na nadpis, nebo v proprties v DisplayName.
3. V toolboxu máme sekci MyActivityLibrary, kde bychom měli vidět naše aktivity. Obě aktivity si přetáhneme z toolboxu do Sekvence.
4. Klikneme levým tlačítkem myši na MyCodeActivity a v properties je vstupní argument Text a do něj zadáme text „My code activita“.



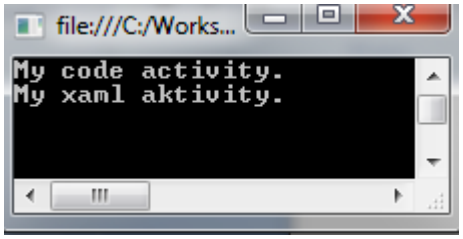
Obr. 3.7 Sekvence realizována v Control flow módu

5. Stiskneme F6 a zkompilujeme projekt.
6. Otevřeme si v designeru workflow1.xaml a z toolboxu přesuneme do hlavního okna Sekvence ze sekce MyActivityLibrary.



Obr. 3.8 Zapojení sekvence do aplikace

7. Kurzorem najedeme na myworkflowConsoleApplication a stiskneme pravé tlačítko myši. Zobrazí se kontextové menu a vybereme Set as StartUp Project.
8. Spustíme projekt stisknutím F5. Na konzolu se nám vypíše text.



Obr. 3.9 Výpis na konzolu

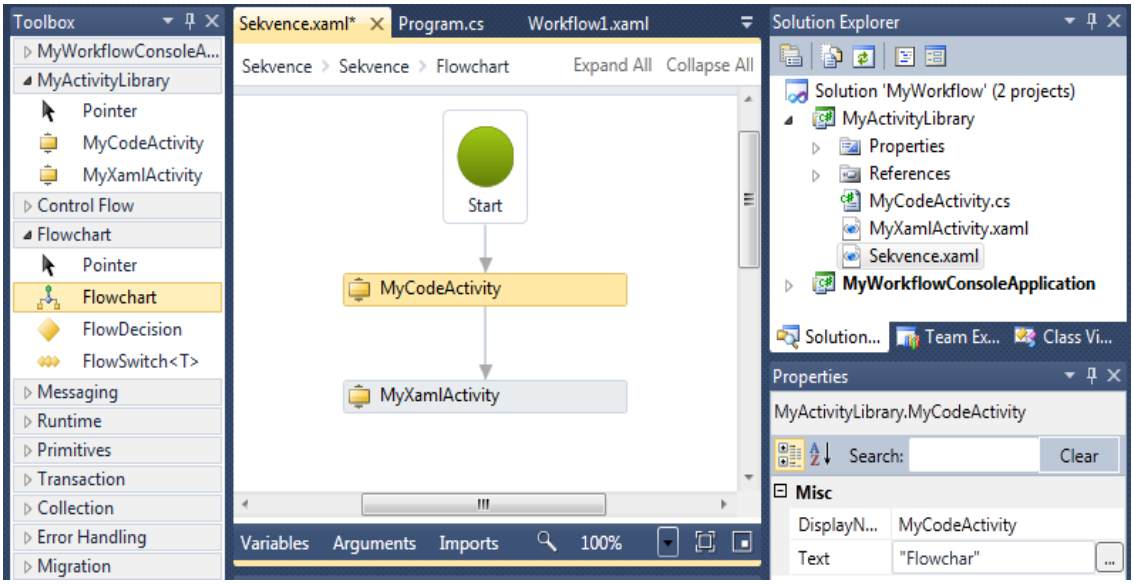
Vytvořili jsme si jednoduchou sekvenci za pomoci control flow a námi vytvořených aktivit. Předvedli jsme si jak vkládat vstupní argumenty do naší aktivity a jak s nimi pracovat a jak z nich vytvářet další aktivity.

Flowchart sekvence

Postup realizace:

1. Z toolboxu ze sekce flowchart přetáhneme komponentu flowchart místo stávající komponenty

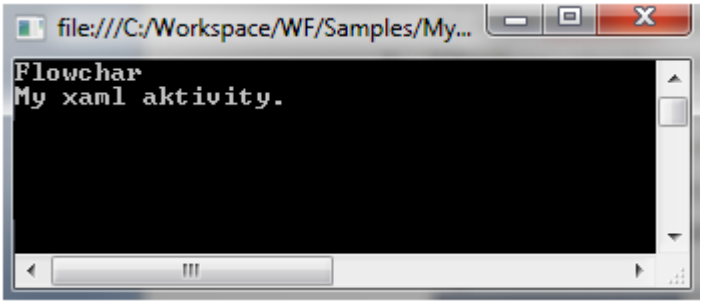
2. Na flowchart klikneme pravým tlačítkem myši a z kontextového menu vybereme Open nebo můžeme flowchart otevřít dvojitým kliknutím levým tlačítkem myši.
3. Z Toolboxu přetáhneme obě naše aktivity do Flowchart.
4. Myší najedeme na prvek start a zobrazí se nám u něj po obvodu malé obdélníčky. Najedeme na jeden z obdélníčků myší, stiskneme levé tlačítko myši a tahem vytvoříme šipku k MyCodeAktivity. Stejným způsobem vytvoříme šipku z MyCodeAktivity k MyXamlAktivity.
5. Známým způsobem zadáme aktivitě MyCodeActivity text „Flowchart.“ do vstupního argumentu Text.



Obr. 3.10 Sekvence realizována v Flowchart módu

6. Otevřeme si v designeru workflow1.xaml a z toolboxu přesuneme do hlavního okna Sekvence ze sekce MyActivityLibrary.

7. Spustíme aplikaci stištěním F5.



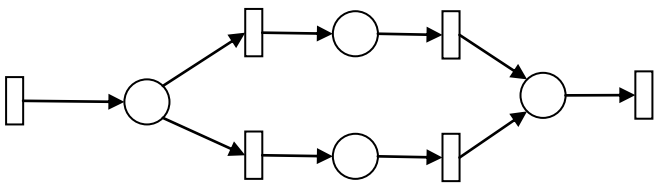
Obr. 3.11 Výstup spuštěné aplikace

Vytvořili jsme stejnou aktivitu jako v předešlém bodu, ovšem k vytvoření jsme použili prvky z Flowchar. Aktivitu vytvořenou pomocí flowchart jsme vložili nakonec předešlé sekvence a workflow jsme spustili. Z výpisu je vidět, že se nejdřív provedla původní část vytvořená pomocí control flow a poté se spustila aktivita, která dělá to samé, vytvořená za pomoci flowchart. Z tohoto příkladu můžeme vidět, že aktivity vytvořené pomocí kontrol flow a flowchart můžeme využívat v obou technikách.

3.2.3 Alternativa

V obou módech v designeru jsou připravené aktivity, které slouží k větvení procesu. V Control Flow je to If a Switch a ve Flowchart je FlowDecision a FlowSwitch. Aktivity If a FlowDecision rozdělují proces na dvě větve. První větev se provede, pokud je podmínka vyhodnocena jako True a druhá jako False. Zbylé dvě aktivity proces rozvětvují na více větví. Pokračuje se tou alternativní cestou, která splňuje zadanou podmínku. Ukážeme si oba typy větvení na jednoduchém příkladu a každý v jiném designérském módu.

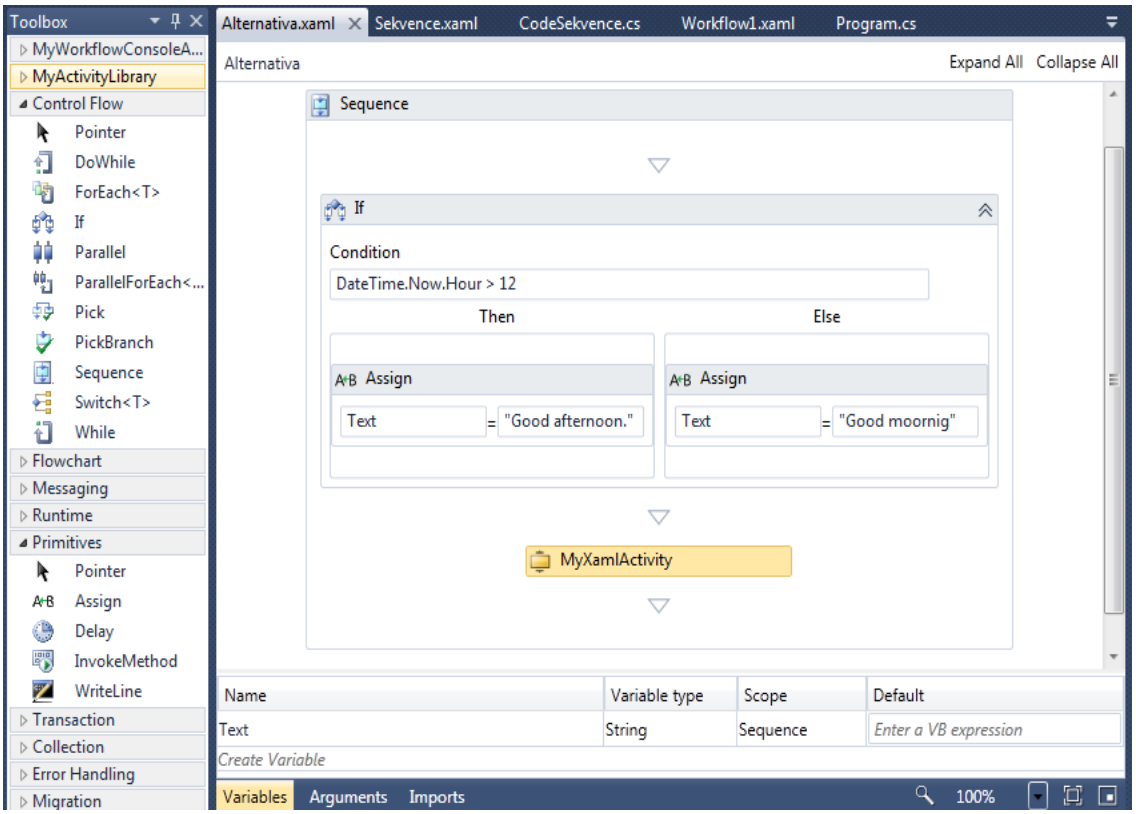
IF v Control flow



Obr. 3.12 Příklad obsahující větvení

Postup realizace:

1. Známým způsobem vytvoříme xaml aktivitu s názvem Alternativa
2. Z toolboxu přetáhneme Sequence a do ní přetáhneme komponentu If. Podmínka bude rozhodovat, o jakou část dne se zrovna jedná. Jestli jde o ráno nebo odpoledne.
3. Vytvoříme si proměnnou Text
4. Do obou větví komponenty if vložíme komponentu Assign, která nám bude přiřazovat zprávu dle aktuální části dne.
5. Do sekvence vložíme pod if naší komponentu MyXamlActivity a do jejího vstupního parametru vložíme proměnnou Text. Stiskneme F6 a zkompilujeme naší knihovnu.

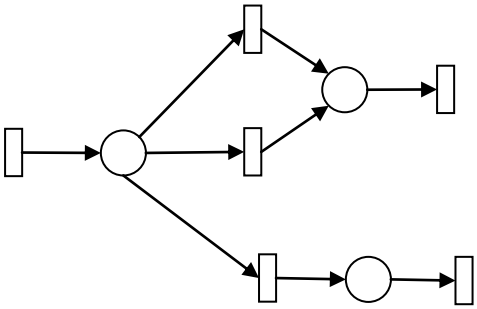


Obr. 3.13 Ukázka využití IF aktivity k větvení procesu

6. V aplikaci smažeme původní knihovnu a z toolboxu přetáhneme Alternativu do hlavního okna. Stiskem tlačítka F5 spustíme aplikaci.

Vytvořili jsme sekvenci, která obsahuje komponentu IF a MyXamlActivitu. IF vyhodnocuje aktuální hodinu. Pokud je aktuální hodina větší než 12, tak se pokračuje větví then, jinak else. Tím se nám proces rozvětvuje na dvě větve. Všimneme si, že pro přiřazení hodnoty proměnné používáme komponentu Assign, kde zadáme název proměnné a hodnotu, kterou do ní vkládáme. Tento proces se dá modelovat i pomocí flowchart, kde se místo komponenty IF použijeme FlowDecision.

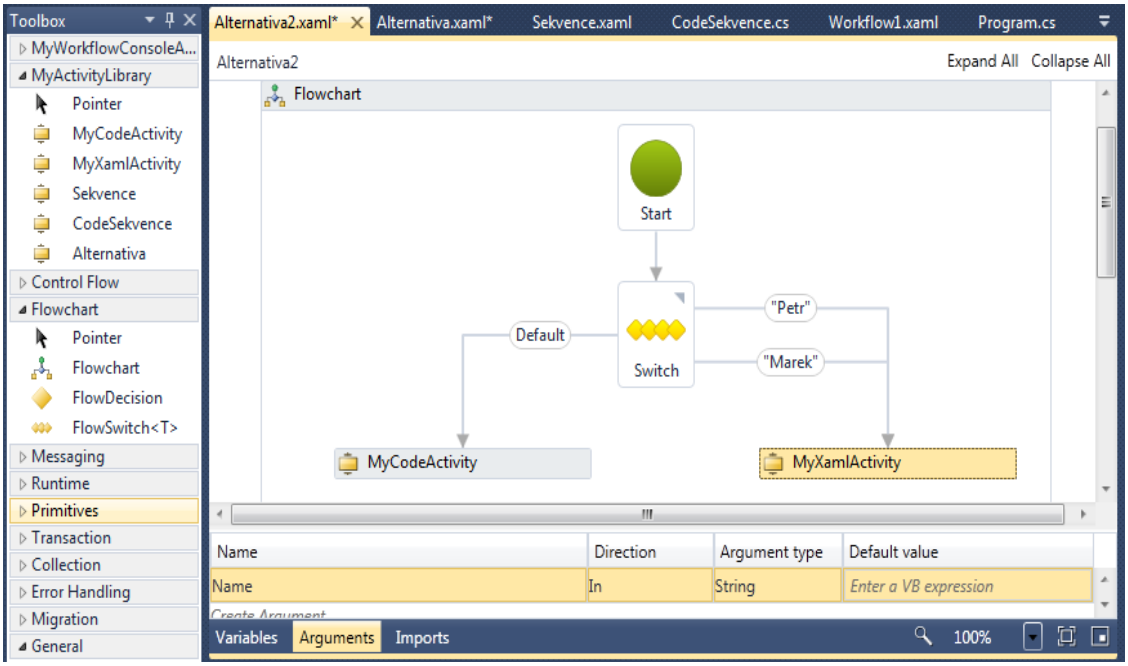
Case pomocí Flowchart



Obr. 3.14 Příkladu obsahující tři větve

Postup realizace:

1. Známým způsobem vytvoříme xaml aktivitu s názvem Alternativa1
2. Z toolboxu přetáhneme Flowchart do hlavního okna.
3. Do flowchart přetáhneme z toolboxu flowswitch<T>, zobrazí se nám okno pro výběr datového typu podmínky. Z nabízených datových typů zvolíme string.
4. Vytvoříme si argument Name.
5. Do podmínky switch vložíme argument Name.
6. Do flowchart vložíme MyCodeActivity, kde vložíme do vstupního argumentu text: „Hello“ +Name a MyXamlActivity, kde bude text “Hello my friend“.
7. Spojíme start s switch, switch s MyCodeActivity a vytvoříme dvě spojení mezi switch a MyXamlActivity. Na první spojení vložíme podmínku „Petr“ a na druhé spojení „Marek“.



Obr. 3.15 Ukázka využití Case k větvení procesu

8. V aplikaci smažeme původní aktivitu a z toolboxu přetáhneme Alternativu1 do hlavního okna a vložíme ji do vstupního parametru name text „Petr“. Spustíme aplikaci.

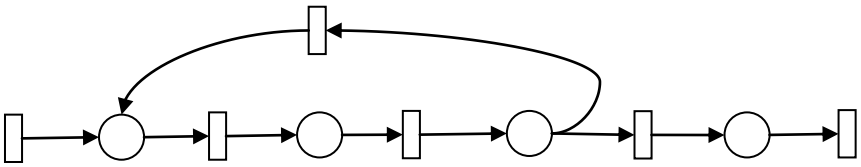
Pomocí komponenty FlowSwitch můžeme proces větvit na více než dvě větve. Komponenta FlowSwitch obsahuje výraz, který je následně vyhodnocen a proces pokračuje větví, která splňuje podmínku. Toto větvení lze realizovat i v Control Flow, kde k tomu slouží komponenta Switch.

3.2.4 Cyklus

Cyklus ve WF4 slouží pro dvě základní věci. Jedna z nich je opakovat činnost, dokud nedosáhneme nějakého stavu procesu. Druhá je procházení nějakého seznamu. První je v Control Flow realizována dvěma komponentami a to While a DoWhile. Obě dvě jsou skoro stejné, činnost je prováděna do té doby, dokud není podmínka splněna. Rozdíl mezi nimi je v tom, že ve verzi DoWhile se aktivita provede alespoň jedenkrát, i když je podmínka splněna už na začátku. V modelování procesu pomocí Flowchart není pro cyklus vytvořena samotná komponenta, která by ho realizovala. Ovšem můžeme ho vytvořit ze sekvence a

alternativy. Tuto realizaci si ukážeme. Pro procházení seznamu má Control Flow vlastní komponentu s názvem ForEach, kterou si taky představíme na příkladu.

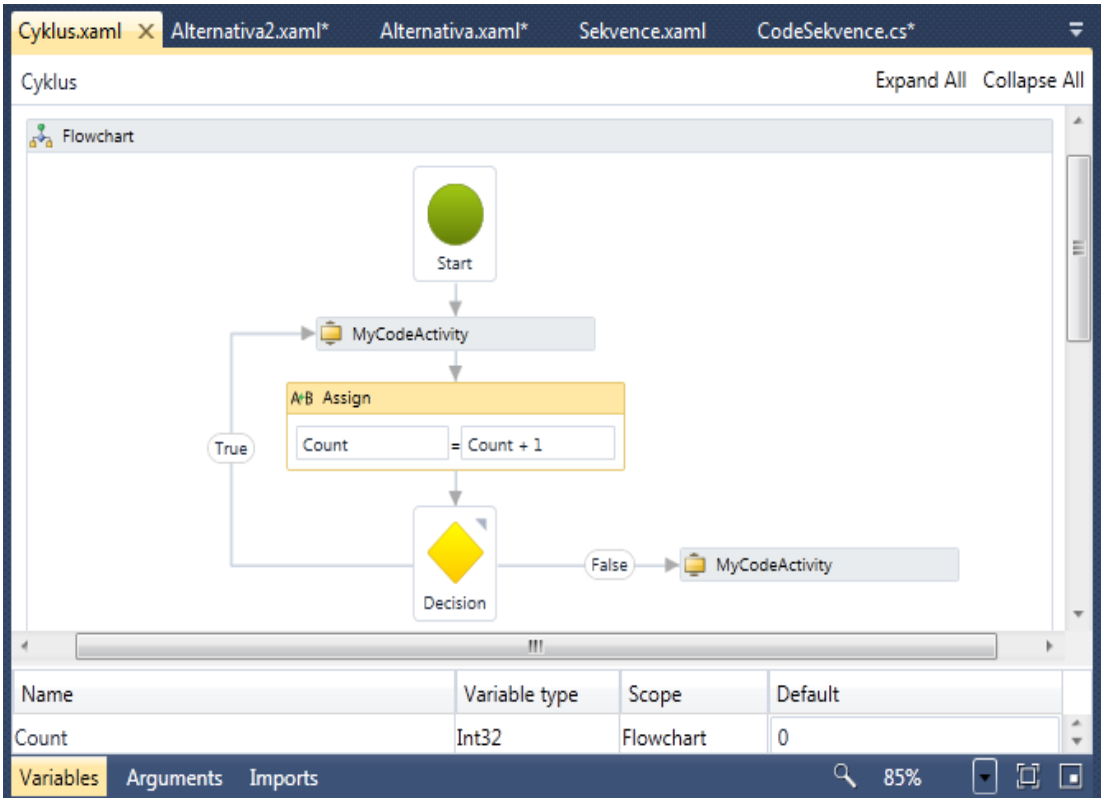
DoWhile pomocí Flowchart



Obr. 3.16 **P**říklad obsahující cyklus

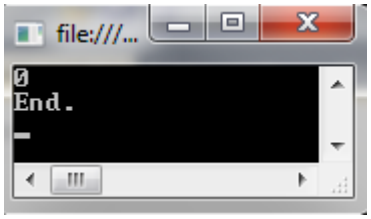
Postup realizace:

1. Vytvoříme známým způsobem Xaml Aktivitu s názvem Cyklus
2. Z toolboxu přetáhneme Flowchart do hlavního okna.
3. Do flowchart přetáhneme z toolboxu dvě MyCodeActivity, Assign a Decision komponenty.
4. Vytvoříme si proměnou Count, která bude datového typu Int32 a s počáteční hodnotou 0.
5. Spojíme komponenty dle obr.1 a v první aktivitě vypíšeme proměnou Count a v poslední aktivitě vypíše řetězec „end“. V komponentě Assign zvýšíme promenou count o jedna a v Decission vložíme podmínku Count<1.



Obr. 3.17 Realizace příkladu s cyklem

6. Zkompilujeme knihovnu. Z aplikace smažeme původní aktivitu a vložíme do ní naší novou a spustíme ji stištěním F5.



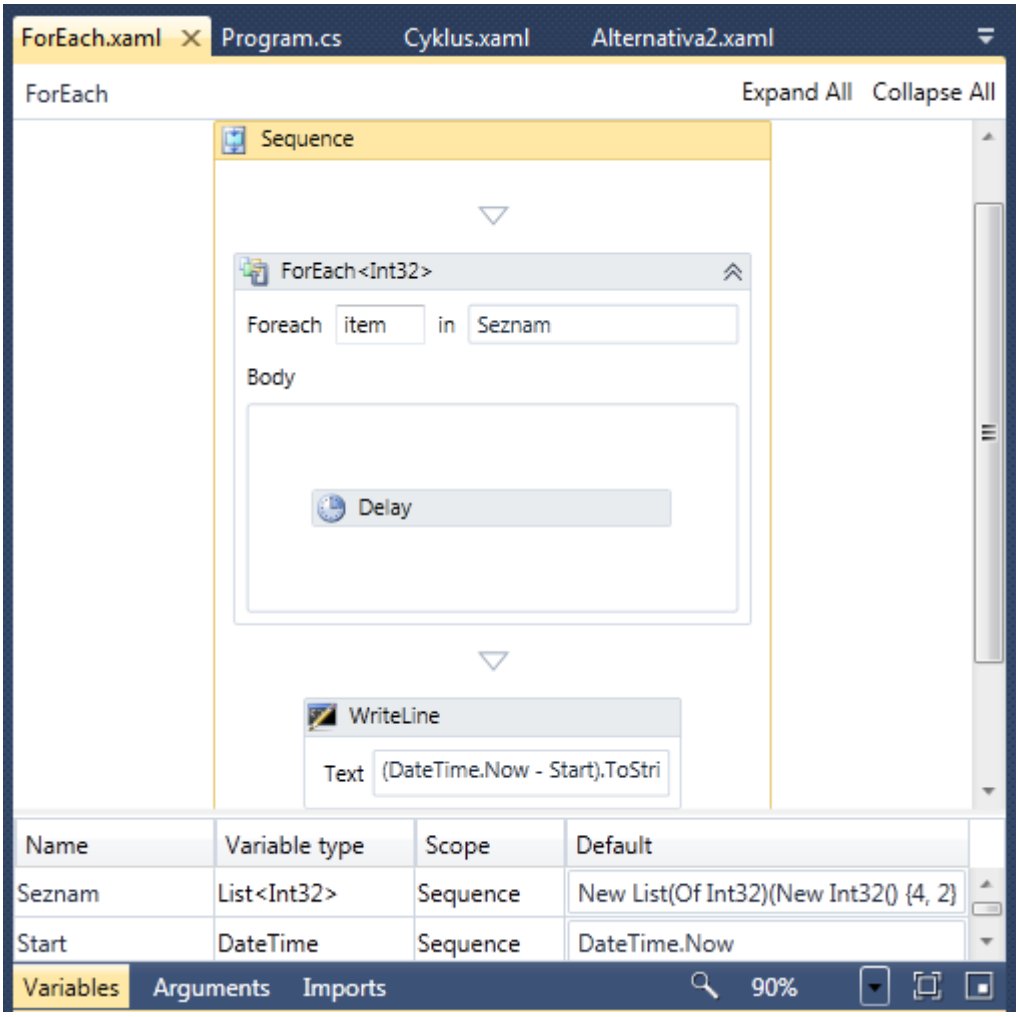
Obr. 3.18 Výstup spuštěné aplikace

Kombinací sekvence a alternativy jsme vytvořili cyklus. Tato realizace je na stejném principu jako komponenta DoWhile v Contorl flow, kde se vždy provede alespoň jedna aktivita. Pokud bychom chtěli realizovat While, stačí přesunout alternativu na začátek procesu, přímo před vykonání aktivity.

Procházení seznamu pomocí ForEach

Postup realizace:

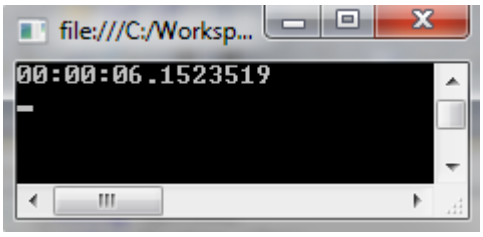
1. Vytvoříme známým způsobem Xaml Aktivitu s názvem ForEach
2. Z toolboxu přetáhneme sequence do hlavního okna.
3. Do sequence si postupně přetáhneme z toolboxu ForEach a WriteLine komponenty
4. Vytvoříme si proměnou Seznam, která bude datového typu List<Int32> Seznam bude obsahovat dvě čísla: 4, 2



Obr. 3.19 Ukázka procházení seznamu v cyklu

5. Vytvoříme si proměnou Start, která bude obsahovat aktuální datum a čas

6. Do komponenty vložíme seznam, který chceme procházet a do body ForEach dáme komponentu Delay. V Delay máme proměnou duration a do ní vložíme TimeSpan.FromSeconds(item)
7. Do komponenty WriteLine vložíme aktuální čas (čas v proměnné Start) Tento převedeme na řetězec
8. Známým způsobem změníme aktivitu a spustíme stisknutím F5



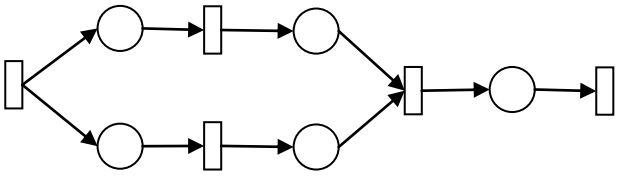
Obr. 3.20 Čas běhu aplikace

Vytvořili jsme seznam, který obsahuje čísla 4 a 2. Tento seznam projdeme pomocí komponenty ForEach. Pro každou iteraci cyklu provedeme spuštění aktivity Delay, která provádí zastavení programu. Aktuální číselná hodnota ze seznamu udává dobu zastavení. Zastavení nám simuluje běh složitější aktivity, abychom si mohli ukázat, jak aktivita ForEach pracuje. Z výstupu je vidět, že prvky se zpracovávali postupně, toho využijeme k porovnání při paralelním zpracování procesu.

3.2.5 Paralelismus

Pro realizaci paralelního běhu procesu má Control Flow dvě komponenty. Jedna z nich se nazývá Paralel a pomocí ni můžeme provádět několik procesů souběžně. Všechny procesy běží v samotném vlákně. Druhou je ParalelForEach, která umožňuje procházet seznam, a pro každý prvek vytvoří samostatné vlákno, ve kterém provede aktivity. Tyto komponenty můžeme použít, pokud prvky a následné zpracování nemají mezi sebou nějaký vztah. Pokud ovšem musíme z nějakého důvodu čekat na dokončení zpracování prvního prvku, abychom mohli zpracovat další, tak je nemůžeme použít. Nespornou výhodou využití paralelismu je urychlení realizovaného procesu. Flowchart nemá vlastní komponentu pro realizaci paralelního běhu, ale můžeme využít komponenty z Control Flow.

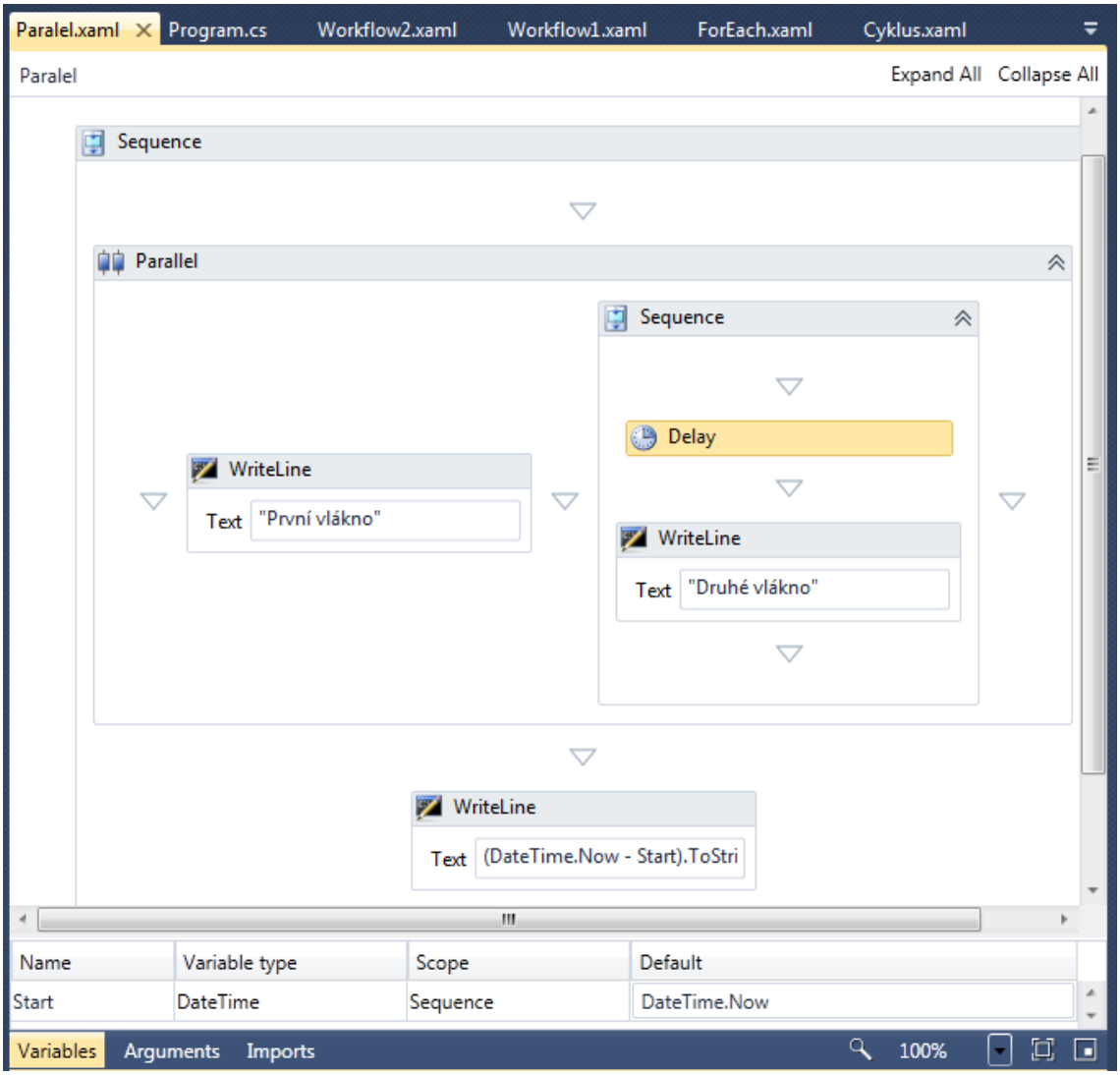
Paralel



Obr. 3.21 Příklad se synchronním paralelismem

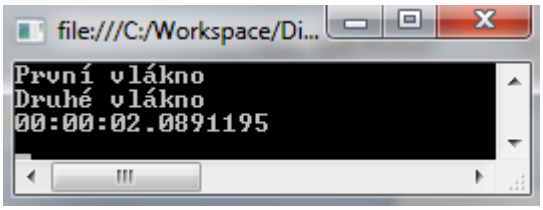
Postup realizace:

1. Vytvoříme známým způsobem Xaml Aktivitu s názvem Paralel
2. Z toolboxu přetáhneme komponentu Sequence a do ní postupně vložíme komponenty Paralel a pod ní WriteLine. Vytvoříme si proměnnou start a nastavíme ji aktuálním datem.
3. Do komponenty Parallel vložíme vedle sebe komponenty WriteLine a Sequence.
4. Do komponenty WriteLine vložíme text: „První vlákno“.
5. Do Sequence vložíme komponenty Delay a WriteLine. V komponentě Delay nastavíme čas na 2s a do WriteLine vložíme text: „Druhé vlákno“.
6. Na závěr vypíšeme čas, který nám trval pro vykonání workflow.



Obr. 3.22 Realizace příkladu obsahující synchronní paralelismus

7. Známým způsobem změníme aktivitu a spustíme stisknutím F5



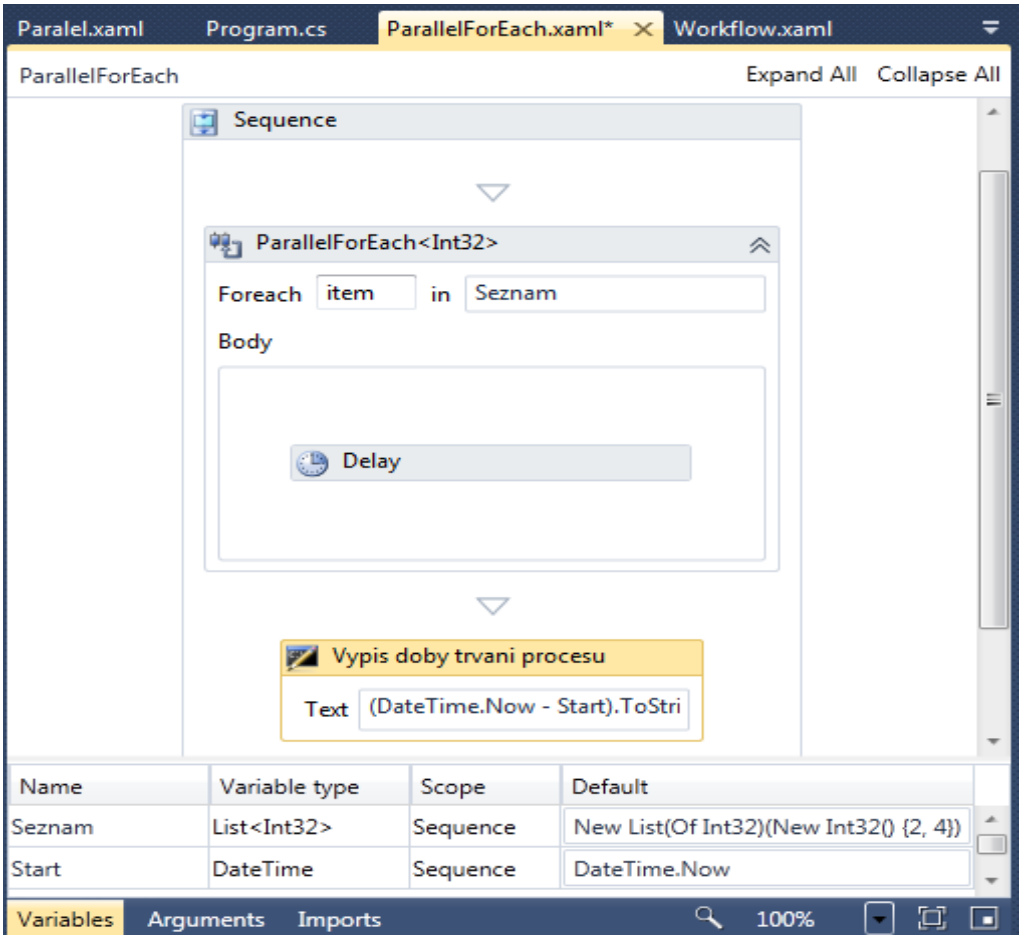
Obr. 3.23 Výpis času běhu aplikace

Z výpisu je patrné, že jde o synchronní paralelismus, protože výsledný čas je vypsán až po provedení všech větví. Pokud by to byl asynchronní paralelismů, tak výpis by byl proveden hned po dokončení jedné z větví.

Procházení seznamu pomocí ParalelForEach

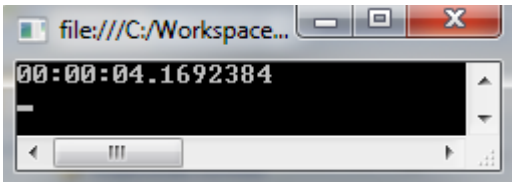
Postup realizace:

1. Vytvoříme známým způsobem Xaml Aktivitu s názvem ParalelForEach
2. Z toolboxu přetáhneme komponentu Sequence a do ní postupně vložíme komponenty ParalelForEach a WriteLine.
3. Dále postupujeme stejně jako u ForEach, vytvoříme proměnné Seznam a Start.
4. Do paralelForEach vložíme k procházení Seznam a do těla vložíme komponentu delay, která bude čekat stejně jako v případě ForEach.
5. Na závěr vypíšeme čas, který nám trval pro vykonání workflow.



Obr. 3.24 Paralelní procházení seznamu

6. Známým způsobem workflow spustíme.



Obr. 3.25 Výpis času běhu aplikace

V předchozí části jsme vytvořili stejné workflow, jen jsme použili komponentu ForEach a nyní ParalelForEach . Rozdíl mezi nimi je zřejmý, u paralelForEach se zpracování vykonávalo souběžně a tím pádem i rychleji. ParalelForEach je synchronní paralelismus, protože jsme spustili dvě vlákna a každé mělo čekat zadanou dobu, a poté se vypíše celkový čas běhu procesu. První vlákno čekalo 2 sekundy a druhé 4 sekundy. Pokud by to byl asynchronní paralelismus, tak by se po doběhnutí prvního vlákna vypsal výsledek něco málo přes 2 sekundy. Ovšem výsledek je 4 sekundy, takže první vlákno čekalo na dokončení druhého a poté se přešlo k provedení poslední aktivity.

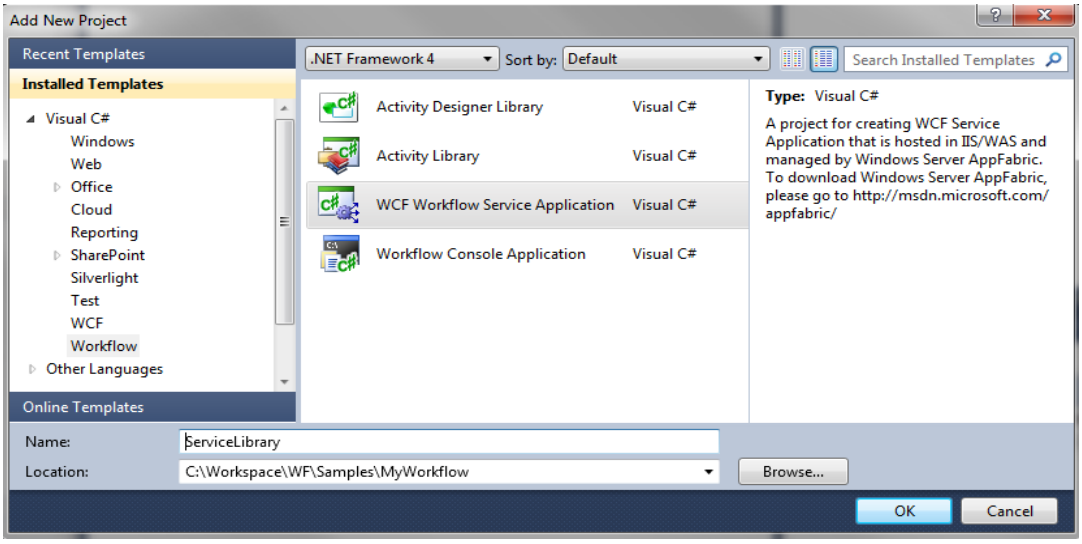
3.3 Využití WCF ve WF 4.0

Nezbytnou součástí dnešních aplikací je možnost komunikovat i s jinými systémy. Pro tento účel existuje v .NET Frameworku technologie Windows Communication Foundation (WCF). WF 4.0 využívá této technologie a umožňuje komunikovat s okolními systémy.WF 4.0 v sobě obsahuje komponentu, která využívá tuto technologii a ukážeme si její použití na jednoduchém příkladě.

Vytvoření Workflow Servisy a využití ve workflow.

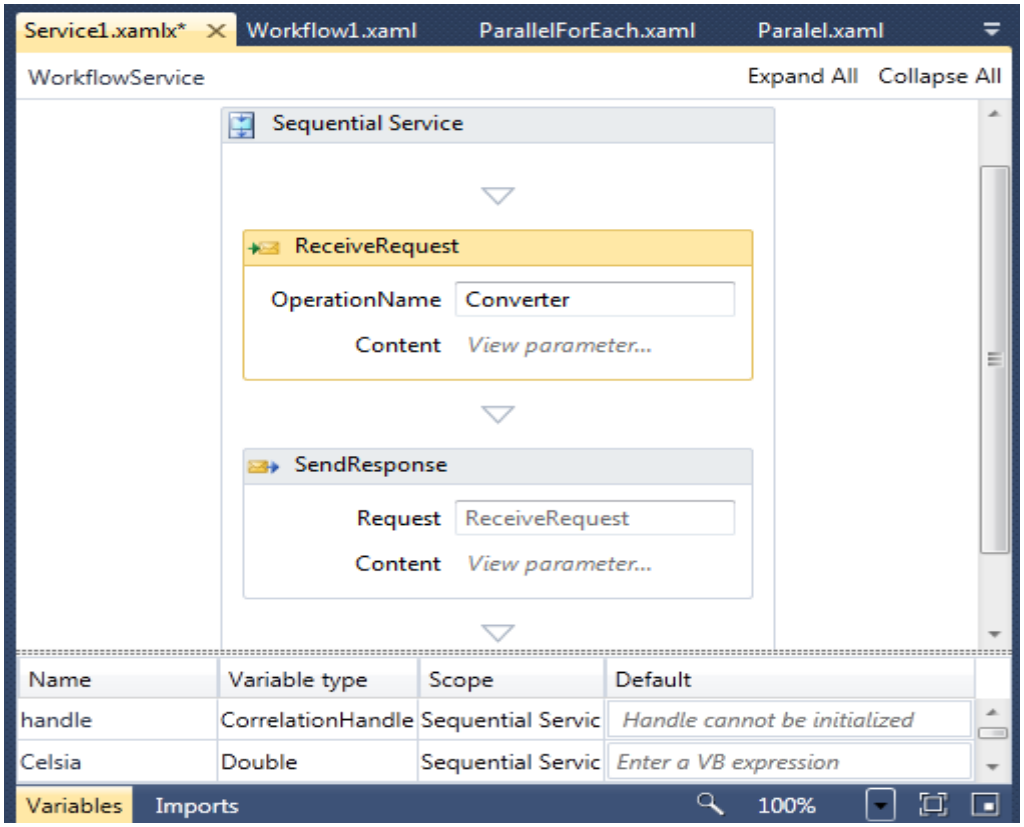
Postup realizace:

1. Stejným způsobem jako jsme přidali MyLibrary přidáme WCF Workflow Service Application s názvem ServiceLibrary.



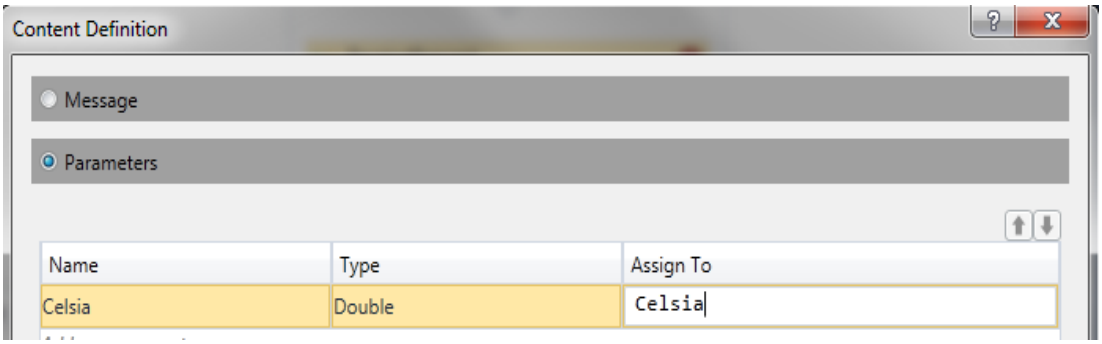
Obr. 3.26 Výběr šablony pro aplikaci obsahující servisu

2. Klikem aktivujeme Sequential Service a otevřeme Variables. Máme před vytvořené dvě proměnné: handle a data. Data změníme na Celsia a přiřadíme datový typ double. Změníme OperationName na Converter.



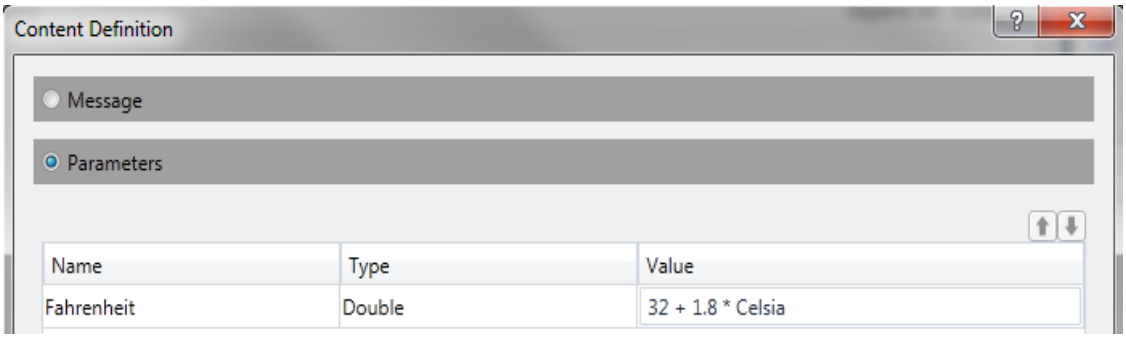
Obr. 3.27 Servisa

3. Dvojklikem na view message v ReceiveReques otevřeme Conten Definition. V tomto okně v sekci Message v textboxu Message data smažeme data. Překlikneme do sekce Parameters a vytvoříme parametr Celsia dle obr.2



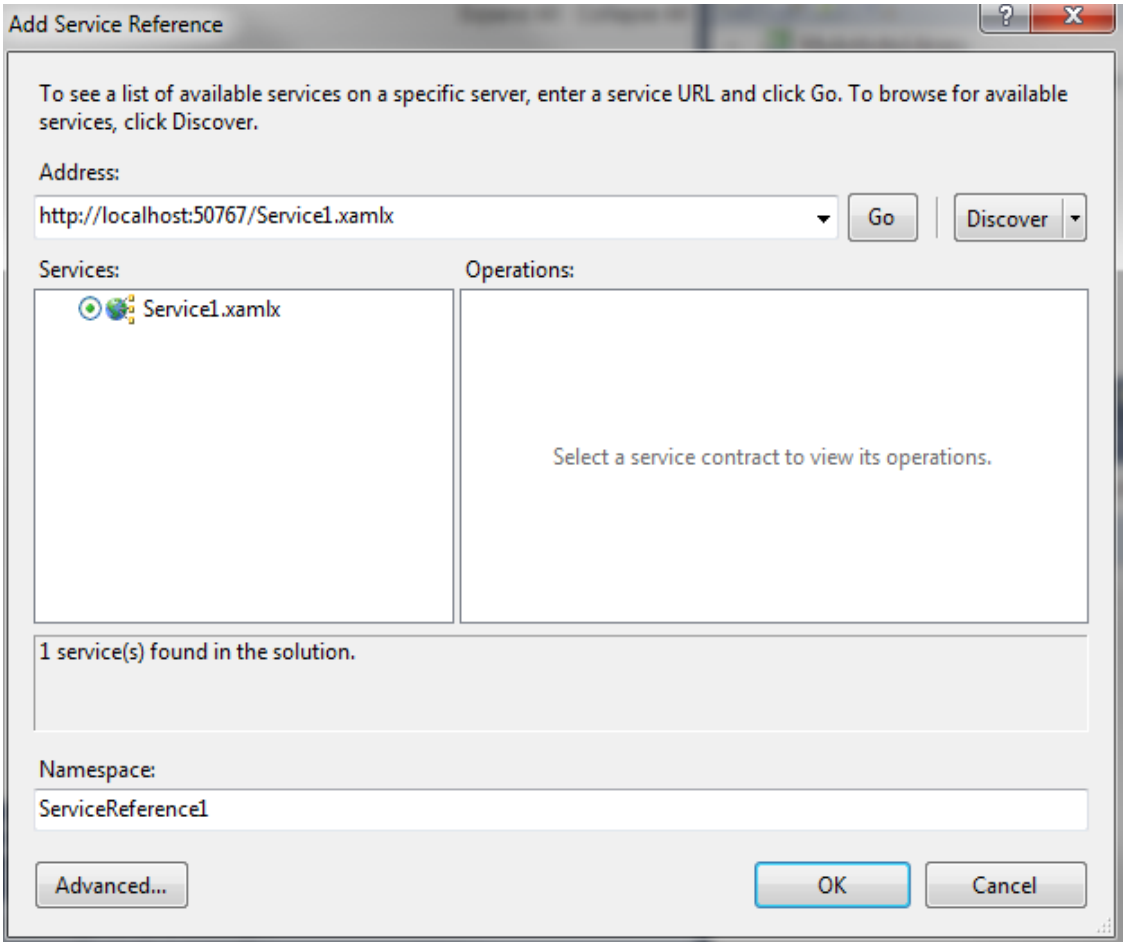
Obr. 3.28 Dialog pro definování vstupu

4. Dvojklikem na view message v SendResponse otevřeme Conten Definition. V tomto okně v sekci Message v textboxu Message data vše smažeme. Překlikneme do sekce Parameter a vytvoříme parametr Fahrenheit dle obr.2. Stiskneme F6 a zkompilujeme ServiceLibrary



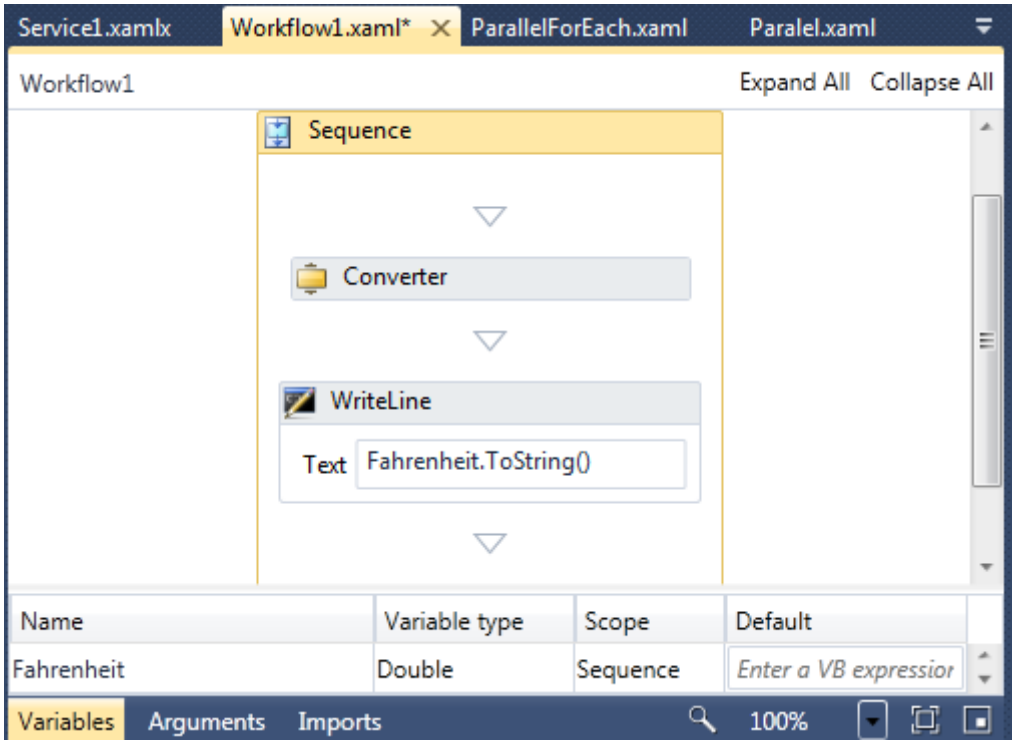
Obr. 3.29 Dialog pro definování výstupu

5. Pravým tlačítkem myši klikneme na MyWorkflowConsoleApplication a z kontextového menu vybereme Add Service Reference
6. Otevře se nám okno Add Service Reference, klikneme na Discovery. Našlo nám to naší servisu a stiskneme OK



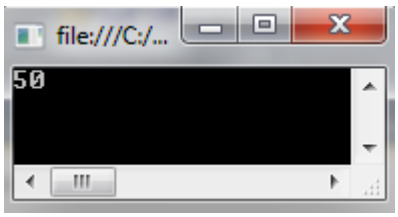
Obr. 3.30 Vytvoření servisy

7. Visual studio nám zobrazí hlášku, že nám, ze servisy vytvoří aktivitu, kterou uvidíme v toolboxu a můžeme ji použít v našem workflow. Pokud ji po spuštění nevidíme v toolboxu, tak stisknutím F6 vše znovu zkompilujeme a už bychom ji měli vidět.
8. Z Workflow smažeme původní aktivitu a přetáhneme do hlavního okna komponentu Sequence. Do ní vložíme Converter a pod Converter přetáhneme komponentu WriteLine.
9. Vytvoříme proměnou Fahrenheit. Klikneme na Converter a v properties nastavíme Celsia na 10 a do Fahrenheit vložíme proměnnou Fahrenheit. Nakonec zobrazíme obsah proměnné Fahrenheit na konsolu.



Obr. 3.31 Zapojení servisy v aplikaci

10. Spustíme stištění F5 aplikaci.



Obr. 3.32 Výpis převodu

Vytvořili jsme si jednoduchou aplikaci, která převádí zadanou hodnotu ve stupních Celsia na stupně Fahrenheit. Tuto službu jsme obalili a používáme ji jako aktivitu.

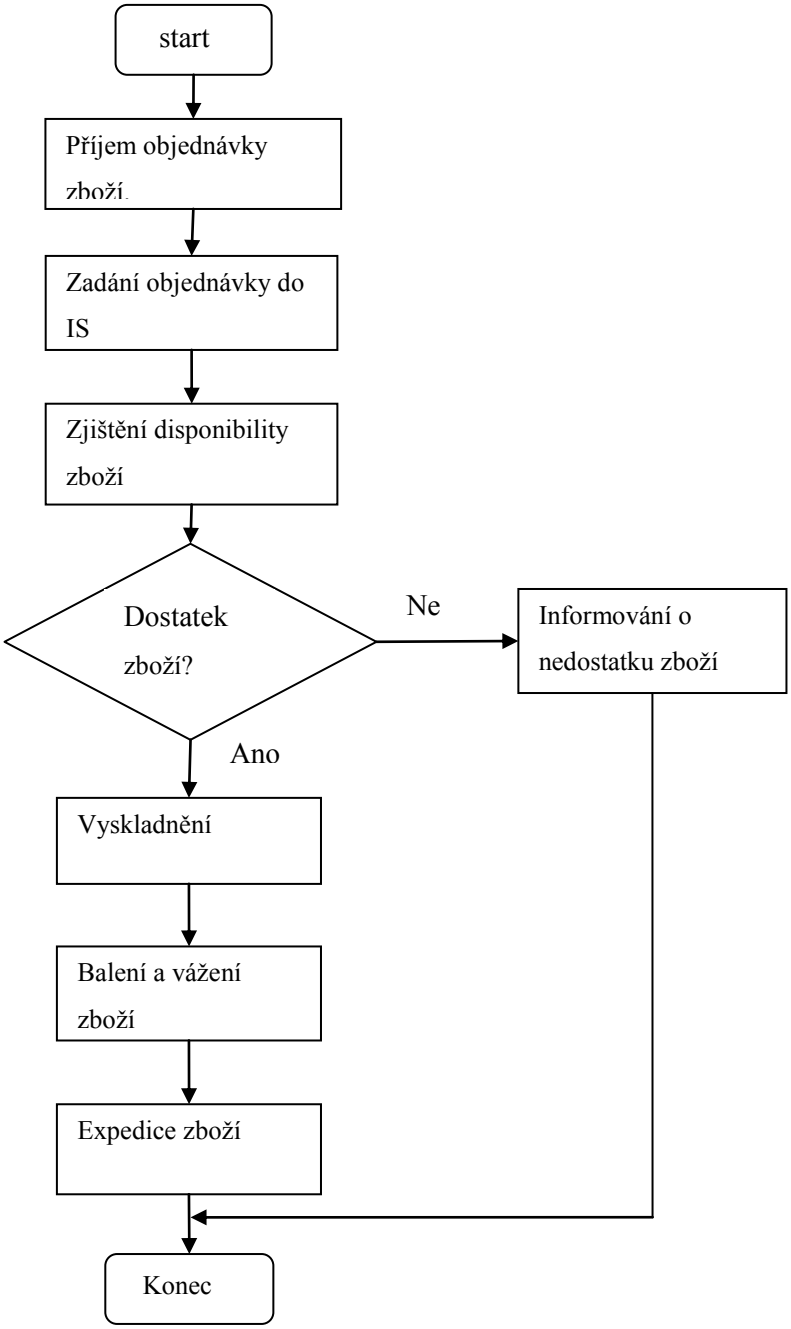
4 Případová studie zvoleného bussines procesu

Společnost CID International, a.s. (dále jen CID) patří mezi přední české producenty informačních systémů pro dopravu, spedici, skladování a CRM. Firma využívá technologii Microsoft .NET framework. Při vývoji informačního systému je kladen důraz na otevřenost systému a modulární architekturu. Otevřenost systému nám umožňuje rozšířit stávající řešení zákazníka a modulární architektura nám poskytuje vysokou míru customizace. V širokém portfoliu produktů firmy CID najdeme jak řešení klient/server, tak i typické desktopové aplikace. Přestože firma v současné chvíli nevyužívá WF 4.0 ve svém řešení, ukážeme si, jeho možné využití při customizaci procesu vyskladnění zboží.

4.1 Kroky procesu vyskladnění zboží

Proces vyskladnění se skládá z několika kroků:

- Příjem objednávky zboží – může být realizován e-mailem, faxem nebo telefonicky.
- Zadání objednávky do IS – jedná se o přepis objednávky do IS.
- Zjištění disponibility zboží – ověření aktuálního stavu na skladě s požadovaným počtem kusů v objednávce.
- Informování o nedostatku zboží – systém odešle požadavek na doplnění zboží do skladu.
- Vyskladnění – představuje úbytek zboží na skladě.
- Balení a vážení – fyzický přesun vybraného zboží k balení a vážení.
- Expedice zboží – zboží čeká k naložení a následné expedici k zákazníkovi.



Obr. 4.1 Proces vyskladnění

Pro ukázku použití WF 4.0 si vybereme proces automatického vyskladnění, který se nachází v těchto činnostech:

- Zjištění disponibility zboží
- Vyskladnění
- Informování o nedostatku zboží

Proces začíná výběrem objednávky zboží a požadavkem k jeho vyskladnění. Objednávka obsahuje seznam zboží a počet kusů. V tomto seznamu se může vyskytovat stejné zboží vícekrát. Toho mohou využívat firmy, které chtějí mít zboží nabalené po určitých kusech. Pro realizaci procesu automatického vyskladnění musí být na skladě dostatek zboží. Pokud je zboží na skladě nedostatek, tak systém o tom informuje uživatele a proces skončí. Vzhledem k tomu, že chybějící zboží je třeba doplnit, tak systém vystaví požadavek na doplnění chybějícího zboží.

V programové realizaci tohoto procesu firmou CID, je tato logika napevno vytvořena v aplikaci. Budeme muset vytvořit vlastní realizaci pomocí WF 4.0, ale před tím musíme provést novou analýzu, která přihlédne k použití této technologie.

4.2 Analýza kroků procesu automatického vyskladnění

Popsali jsme si proces a teď se pokusíme definovat jednotlivé kroky procesu a podrobněji si je přiblížit. Z těchto kroků posléze vytvoříme samostatné aktivity. Aktivity nám ve WF 4.0 budou sloužit jako základní stavební bloky.

4.2.1 Popis aktivit

Výběr objednávky k vyskladnění

Z nabídky objednávek vybereme objednávku, kterou chceme vyskladnit. U objednávky uvedeme datum, na kdy potřebujeme mít vyskladněné zboží. Objednávka obsahuje seznam zboží a jeho počty kusů.

Setřídění zboží na výdejce

Na výdejce se může nacházet více objednávek stejného druhu zboží. Při kontrole disponibility zboží, by mohlo dojít k tomu, že na skladě by byl nedostatek zboží, ale systém by stav zboží ve skladě vyhodnotil jako dostatečný. Tuto situaci si vysvětlíme na příkladu. Na skladě je 8 kusů daného zboží. V seznamu zboží výdejky se nachází dvě položky, na kterých je objednáno po 5 kusech daného zboží (dohromady 10 kusů). Pokud bychom procházeli seznam po položkách, tak bychom vyhodnotili tyto dvě položky jako

dostatečné, i když v součtu je na skladě zboží nedostatek. Proto musíme nejdřív položky na seznamu setřídit a až poté vyhodnotit.

Vytvoření seznamu obsahující chybějící zboží na skladě

Na výdejce se nachází druh zboží a počet kusů, které chceme vyskladnit. Zboží je setříděné a můžeme provést kontrolu stavu skladu s objednávkou. Abychom mohli provést automatické vyskladnění, tak na skladě musí být všechny požadované kusy zboží. Pokud by chyběl třeba jen jeden kus nějakého zboží, tak proces automatického vyskladnění nelze realizovat. V tomto případě se naskýtá možnost informovat o tomto stavu správce skladu, aby mohl požadované zboží objednat.

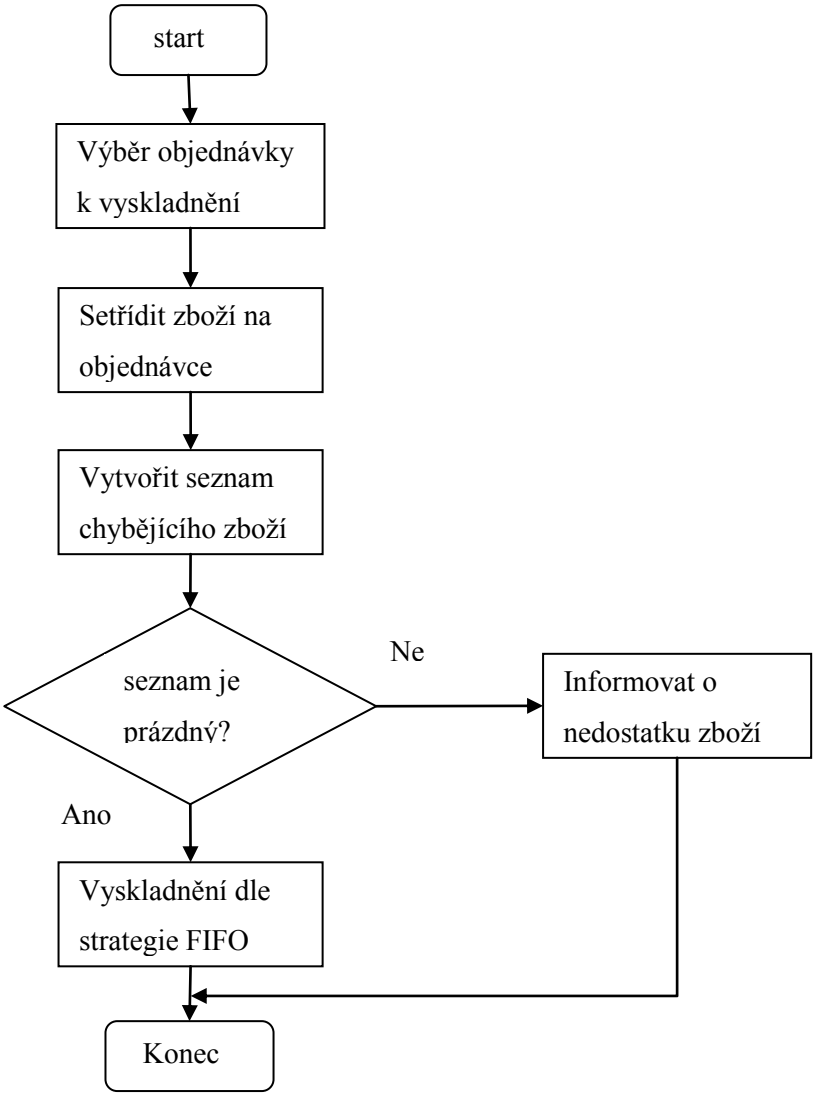
Vyskladnění FIFO

Pokud je na skladě dostatek zboží, tak můžeme zboží vyskladnit. Zboží máme naskladněné v různých balíčcích, v závislosti na tom, jak jsme zboží přijali. Nyní provedeme vyskladnění dle vyskladňovací strategie FIFO:

Zboží se vyskladňuje dle data příjmu zboží. Zboží, které má starší datum příjmu, bude upřednostněno při vyskladnění. (Fronta zboží vytvořená na základě data přijmu zboží od nejstaršího k nejmladšímu).

Odeslání informace o nedostatku zboží na skladě

Při kontrole disponibility zboží na skladě můžeme rovnou vytvořit seznam zboží, které nám chybí. Každá položka seznamu by obsahovala název zboží a počet požadovaných kusů. Tento seznam můžeme poté odeslat systému, který slouží k objednávkám zboží do skladu.



Obr. 4.2 Automatické vyskladnění

Museli jsme vytvořit novou analýzu procesu s přihlédnutím na to, že budeme využívat technologii WF 4.0. V ní jsme museli proces rozdělit na jednotlivé logické kroky, ze kterých budeme vytvářet aktivity.

Firma CID ovšem v krátké době bude své řešení nasazovat u dalších 2 zákazníků. Jejich proces vyskladnění nebude úplně korespondovat s původním řešením firmy CID a musí se provést customizace.

4.3 Analýza odlišnosti procesu automatického vyskladnění u prvního zákazníka

Zákazník vlastní dva sklady, ve kterých bude nasazený systém od firmy CID. V původním řešení procesu automatického vyskladnění budeme muset provést několik úprav:

- Změna vyskladňovací strategie z FIFO na LIFO. Při výběru zboží se vytvoří fronta, které bude setříděna od dřívějšího naskladnění k pozdějšímu.
- Při nedostatku zboží na skladu, požaduje zákazník před objednáním zboží od dodavatele, zkontrolovat stav zboží i ve druhém skladě. Pokud zboží bude ve druhém skladě, tak o tom systém informuje uživatele. Jestliže zboží ve druhém skladě nebude, tak je automaticky vystaven požadavek na objednávku chybějícího zboží.
- V objednávce se nemůže vyskytnout vícekrát zboží stejného druhu.

Z požadavků zákazníka vyplývá, že budeme muset realizovat několik nových aktivit:

Vyskladnění pomocí strategie LIFO

Změna oproti strategiím FIFO je v setřídění zboží. Zboží, které bude uskladněno jako poslední, bude upřednostněno při vyskladnění.

Vyhledat zboží ve druhém skladě

Při nedostatku zboží na skladu, je potřeba před objednáním zboží od dodavatele, prohlédnout i druhý sklad.

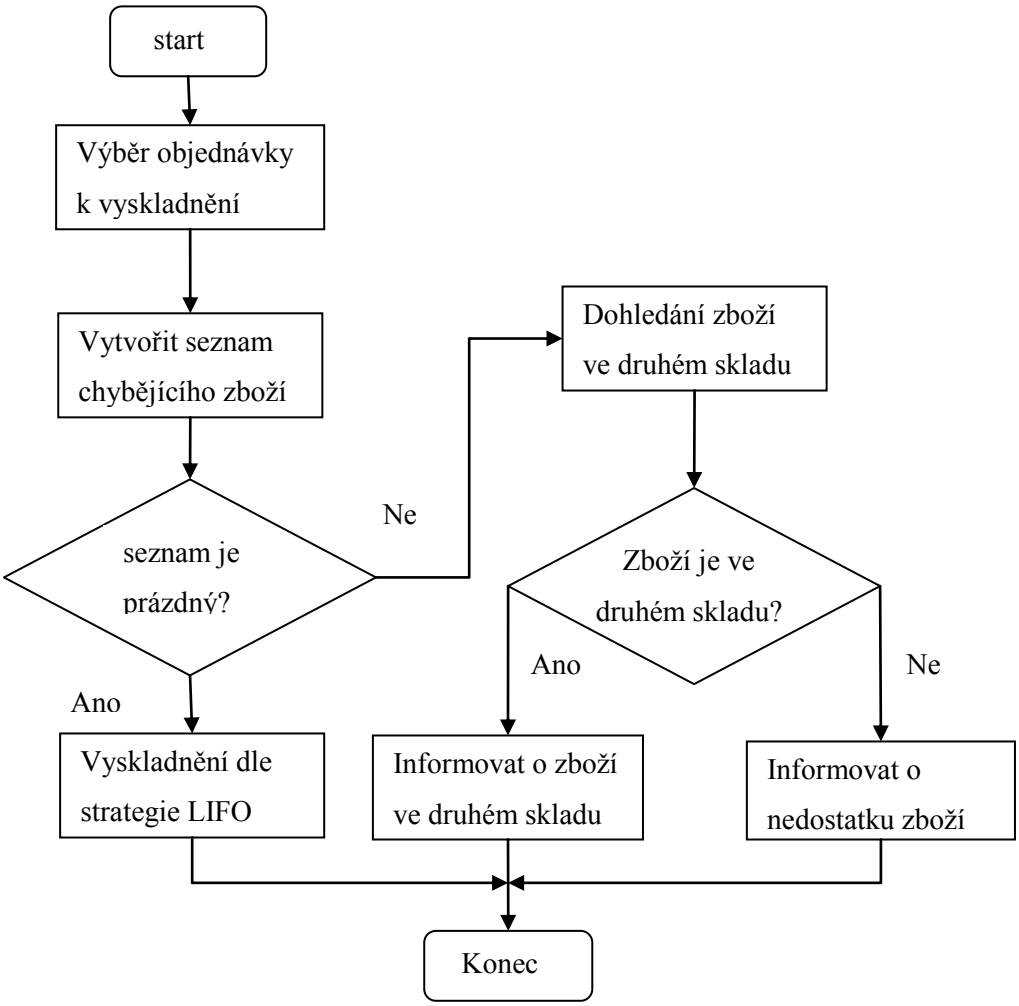
Informovat o zboží ve druhém skladě

Systém vyrozumí uživatele o tom, že zboží se nachází ve druhém skladě.

Nesetřídit zboží

V objednávce se nemůže vyskytnout vícekrát zboží stejného druhu.

Provedli jsme analýzu odlišnosti procesu automatického vyskladnění u prvního zákazníka od původního řešení. Model procesu vyskladnění prvního zákazníka si můžeme prohlédnout na obr. 4.3.



Obr. 4.3 Proces automatického vyskladnění u prvního zákazníka

4.4 Analýza odlišnosti procesu automatického vyskladnění u druhého zákazníka

Pro druhého zákazníka, který má malý sklad na ovoce, musíme proces vyskladnění tomu přizpůsobit:

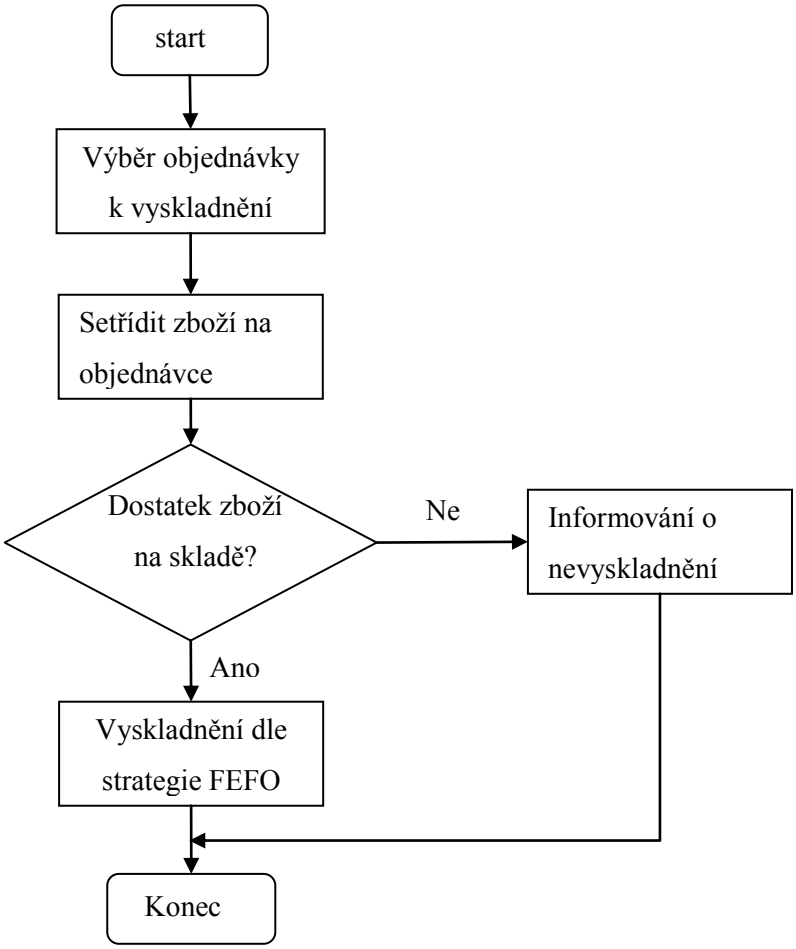
- Nebudeme zakládat požadavek na objednávku chybějícího zboží, ale jen informujeme uživatele o tom, že se nepodařilo vyskladnit zboží.
- Z toho vyplývá, že není nutné vytvářet seznam chybějícího zboží.
- Jelikož zboží na skladě je závislé na datu spotřeby, tak pozměníme vyskladňovací strategii z FIFO na FEFO (First expire first out). Při výběru zboží se vytvoří fronta, která bude setříděna od dřívějšího data exspirace k pozdějšímu.

Z analýzy vyplývá, že budeme muset realizovat novou aktivitu:

Vyskladnění FEFO

Strategie FEFO je stejná jako FIFO, s tím rozdílem, že sledujeme místo data příjmu na sklad, datum exspirace zboží.

Provedli jsme analýzu odlišnosti procesu automatického vyskladnění u druhého zákazníka od původního řešení. Jak vypadá proces vyskladnění u druhého zákazníka, si můžeme prohlédnout na obr. 4.4.



Obr. 4.4 Proces automatického vyskladnění u druhého zákazníka

5 Implementace případové studie

V předchozí kapitole jsme si uvedli proces automatického vyskladnění. Vytvořili jsme si analýzu jednotlivých aktivit, které v této části budeme realizovat. Z těchto aktivit složíme požadovaný proces.

Ukážeme si i customizaci existujícího řešení pro nové zákazníky, přičemž vyjdeme z naší předchozí analýzy odlišností u jednotlivých zákazníků. Rovněž si ukážeme výhody při použití WF 4.0.

5.1 Proces automatického vyskladnění

Nejdříve si musíme vybrat objednávku, kterou chceme vyskladnit, a poté proces spustíme.

Vstup

- String ConnectionString (řetězec k napojení na DB)
- Int32 lIDPriVydej (primární klíč objednávky, kterou chceme vyskladnit)

Výstup

- String Message (vyrozumění uživatele o výsledku procesu)

5.1.1 Jednotlivé aktivity

Vytvoříme si jednotlivé aktivity, u kterých nás nejvíce zajímá jejich rozhraní, abychom je mohli mezi sebou propojit a realizovat tak proces automatického vyskladnění.

SetřiditZbozi

Vstup – List<PriPolozVydej> (seznam zboží, které chceme vyskladnit)

Výstup – Dictionary<int,int> (slovník, kde klíčem je IDZbozi a hodnotou jsou kusy zboží)

Ze zadaného seznamu vytvoří „slovník“, který neobsahuje duplicity stejného zboží.

Vytvoření seznamu obsahující chybějící zboží na skladě

Vstup

- Dictionary<int,int> mapaZbozi (slovník, kde klíčem je IDZbozi a hodnotou jsou kusy zboží)
- VydejkaDAO DB (slouží k přístupu k databázi)

Výstup

- Dictionary<int,int> ObjednavkaZbozi (slovník, kde klíčem je IDZbozi a hodnotou jsou chybějící kusy zboží)

Ze zadaného „slovníku“ vytvoří nový „slovník“, ve kterém je uveden druh zboží a počet chybějících kusů.

FIFO

Vstup

- List<PriPolozVydej> PriPolozVydej (seznam zboží, které chceme vyskladnit)
- PriVydej PriVydej ()
- VydejkaDAO DB (slouží k přístupu k databázi)

Provede odečet zboží ze skladu dle strategie FIFO.

Import

Vstup

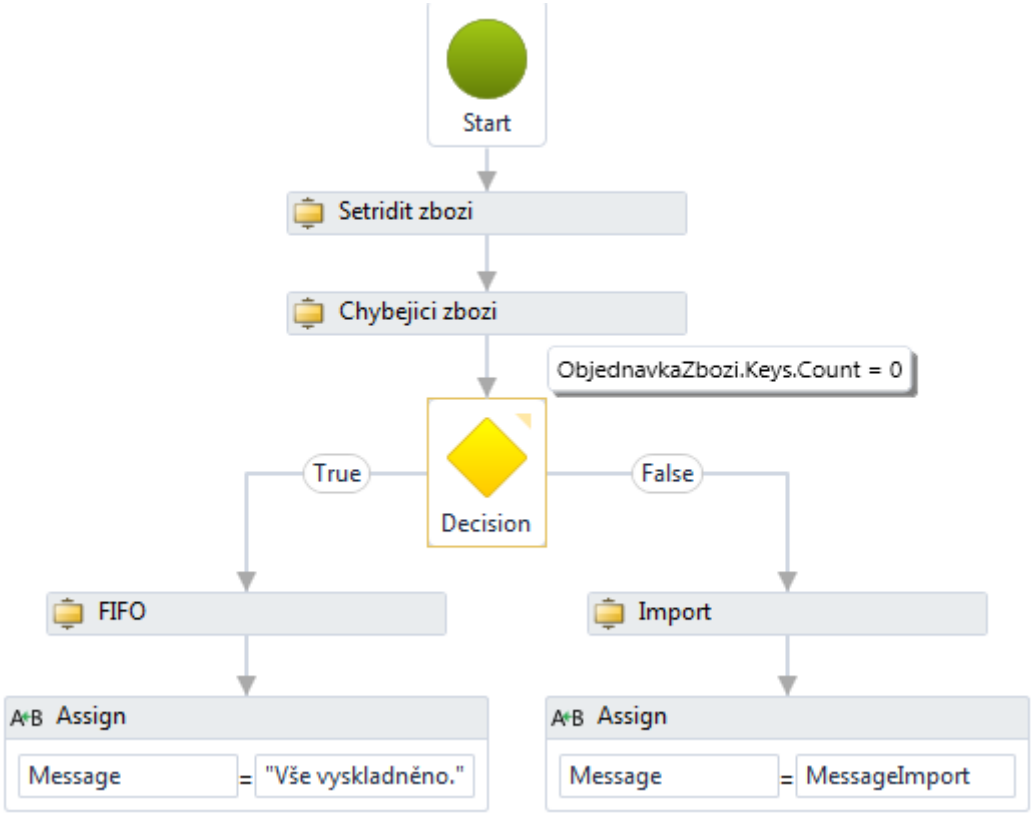
- Dictionary<int,int> ObjednavkaZbozi (slovník, kde klíčem je IDZbozi a hodnotou jsou chybějící kusy zboží)
- VydejkaDAO DB (slouží k přístupu k databázi)
- PriVydej PriVydej (objednávka zboží)

Výstup

- String Message (Výsledek založení požadavku na import zboží)

Odeslání informace o nedostatku zboží na skladě.

5.1.2 Postup řešení



Obr. 5.1 Grafický náhled na implementaci procesu automatického vyskladnění

Proces vyskladnění začíná aktivitou *Setrid zbozi*, která ze zadaného seznamu zboží, vytvoří setříděný seznam. Tento setříděný seznam je vstupem pro aktivitu *Chybejici zbozi*, která ověří dostupnost zboží na skladě. Pokud nějaké zboží chybí, tak ho vloží do nového seznamu (*ObjednavkaZbozi*) a uvede u něj i počet chybějících kusů. Dalším krokem se nám proces větví. Pokud v seznamu chybějícího zboží není žádný záznam, tak se provede aktivita FIFO, která zboží vyskladní, a poté o tom informujeme uživatele. Pokud ovšem zboží na skladě chybí, tak se provede aktivita *import*, která zadá požadavek na objednání nového zboží na sklad, dle seznamu s chybějícím zbožím. Nakonec je uživatel informován o nevyskladnění zboží a založení požadavku na import zboží.

5.2 Customizace procesu pro prvního zákazníka

Připomeňme si odlišnosti procesu od původního řešení:

- Změna vyskladňovací strategie z FIFO na LIFO
- Vyhledání chybějícího zboží ve druhém skladu
- Nemusíme zboží setřiďovat
- Vyrozumět uživatele o dostatečném počtu požadovaného zboží ve druhém skladu.

Máme provést celkem čtyři úpravy a vytvářet budeme jen dvě nové aktivity. To proto, že na vyrozumění použijeme již vytvořenou aktivitu s názvem *assign* a pomocí ní vložíme do výstupní zprávy procesu vyrozumění o požadovaném počtu zboží ve druhém skladu. Už nám zbývá jen jedna úprava, a to, že nemusíme mít zboží setříděné. Ovšem pro aktivitu s názvem *Chybejici zbozi* potřebujeme mít seznam, kde bude uvedeno v záznamu jen typ zboží a počet kusů. Naskýtá se nám dvě možnosti, buď vytvořit novou aktivitu, která ze vstupních dat vytvoří požadovaný seznam, nebo využít aktivitu *Setridit Zbozi*, která už toto dělá. My si vybereme druhou možnost, abychom nevytvářeli zbytečné novou aktivitu.

5.2.1 Nové aktivity

Vyhledat zboží ve druhém skladě

Vstup

- Dictionary<int,int> ZboziKS (slovník, kde klíčem je IDZbozi a hodnotou jsou chybějící kusy zboží)
- VydejkaDAO DB (slouží k přístupu k databázi)

Výstup

- Boolean ObsahujeVsechnoZbozi (pokud druhý sklad obsahuje všechno chybějící zboží, tak vrátí true, jinak false)

Pomocí WCF se dotážeme na chybějící zboží ve druhém skladě.

LIFO

Vstup

- List<PriPolozVydej> PriPolozVydej (seznam zboží, které chceme vyskladnit)
- PriVydej PriVydej (objednávka zboží)
- VydejkaDAO DB (slouží k přístupu k databázi)

Provede odečet zboží ze skladu dle strategie LIFO.

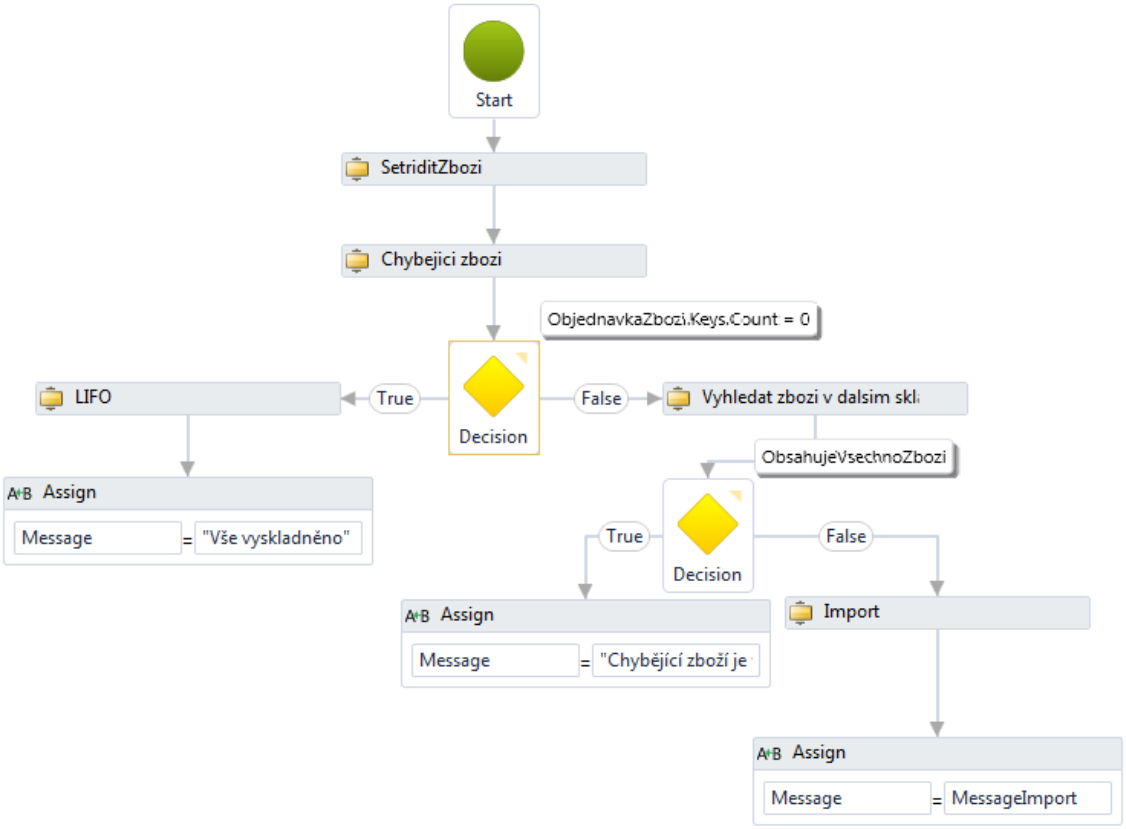
5.2.2 Postup úpravy původního řešení

Vytvoříme si kopii původního řešení, kterou budeme přetvářet pro nového zákazníka. První úpravu provedeme změnou vyskladňovací strategie z FIFO na LIFO. Co vše pro to musíme udělat:

- Odstranit aktivitu FIFO
- Vložit aktivitu LIFO a napojit ji vodícími čárami
- Vložit do aktivity vstupní parametry

Druhou úpravu provedeme před objednávkou chybějícího zboží. Musíme rozšířit větev, ve které zakládáme požadavek na import nového zboží, o dohledání zboží ve druhém skladě.

- Vložíme aktivitu *Vyhledat zbozi v dalsim skl* před aktivitu *Import*
- Zadáme do nové aktivity vstupní parametry a uložíme si její výstupní parametr
- Za ní vložíme rozhodovací blok, ve kterém vyhodnotíme výstupní parametr nové aktivity
- Do větve true napojíme aktivitu *Import* a zbytek původní větve
- Ve větvi false vyrozumíme uživatele o zboží ve druhém skladě
- Všechny aktivity spojíme vodícími linkami v požadovaném pořadí



Obr. 5.2 Grafický náhled na implementaci procesu u prvního zákazníka

5.3 Customizace procesu pro druhého zákazníka

Připomeneme si odlišnosti procesu od původního řešení:

- Nechceme zakládat požadavek na import, pokud je nedostatek zboží na skladě, ale jen podat informaci o tom, že se nepodařilo vyskladnit zboží.
- Z toho vyplývá, že nepotřebujeme vytvářet ani seznam chybějícího zboží.
- Jelikož zboží na skladě je závislé na datu spotřeby, tak požaduje změnu vyskladňovací strategie z FIFO na FEFO (First expire first out). Při výběru zboží se vytvoří fronta, které bude setříděna od dřívějšího data expirace k pozdějšímu.

Máme provést tři úpravy. Celou větve, ve které se nachází aktivita *Import*, nahradíme aktivitou *Assign*, pomocí které vyrozumíme uživatele nevyskladnění zboží. Ačkoliv nemusíme vytvářet seznam s chybějícím zbožím, tak i přes to využijeme aktivit *Setrid Zbozi* a *Chybejici zbozi* a to abychom mohli rozhodnout, jestli je dostatek zboží na skladu. Pro poslední úpravu musíme vytvořit novou aktivitu, která provede vyskladnění dle strategie FEFO.

5.3.1 Nové aktivity

FEFO

Vstup

- List<PriPolozVydej> PriPolozVydej (seznam zboží, které chceme vyskladnit)
- PriVydej PriVydej (objednávka zboží)
- VydejkaDAO DB (slouží k přístupu k databázi)

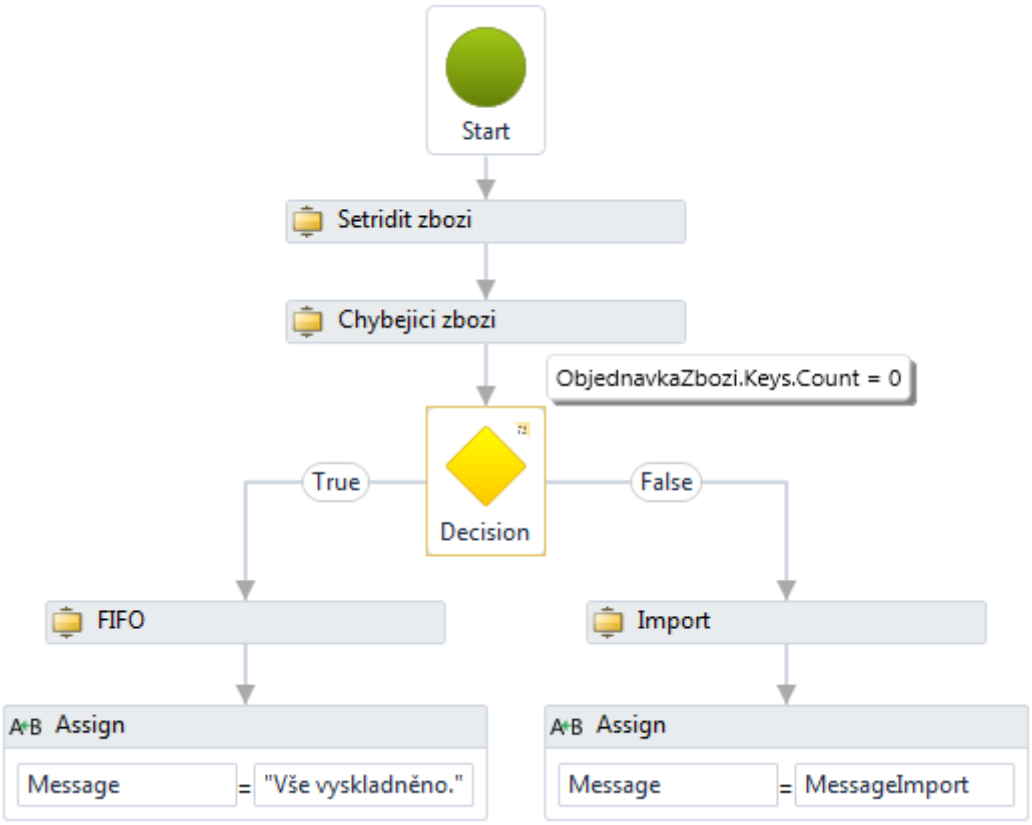
Provede odečet zboží ze skladu dle strategie FEFO.

5.3.2 Postup úpravy původního řešení

Vytvoříme si kopii původního řešení, kterou budeme přetvářet pro nového zákazníka. První úpravu provedeme změnou vyskladňovací strategie z FIFO na FEFO. Co vše pro to musíme udělat, jsme si uvedli v úpravě pro prvního zákazníka.

Ve druhé úpravě musíme vyrozumět uživatele, že proces automatického vyskladnění skončil neúspěšně z důvodu nedostatku zboží.

- Odstraníme větev procesu, ve které je obsažena aktivita *Import*
- Vložíme aktivitou *Assign* místo původní větve
- Natavíme zprávu uživateli pomoci aktivity *Assign*



Obr. 5.3 Grafický náhled na implementaci procesu u druhého zákazníka

5.4 Zhodnocení customizace

Pokud bychom řešili customizaci klasickou metodou, tak bychom pro každého zákazníka museli provádět nové řešení úplně od začátku. Jelikož jsme využili technologii WF 4.0, tak se nám povedlo použít části původního procesu, které se shodovali s novým procesem. Odlišnosti procesů jsme řešili vytvořením nových aktivit a zakomponováním do stávajícího řešení. Tím jsme dosáhli časové úspory, protože jsme nemuseli realizovat celý nový proces od začátku.

6 Závěr

Ukázali jsme si základní techniky modelování bussines procesů a seznámili jsme se s možnostmi WF 4.0. Prokázali jsme, že jakýkoliv bussines proces jsme schopni realizovat pomocí WF 4.0, neboť i nestandardní bussines model dokážeme realizovat vlastní aktivitou. Na vybraném podnikovém procesu jsme si ukázali, jak takovýto proces implementovat pomocí této technologie.

Bezesporu velkou výhodou WF 4.0. je grafická reprezentace. Tento grafický náhled pomáhá rychleji pochopit logiku procesu, což může výrazně přispět k zrychlení pochopení převzatých řešení. U rozsáhlých procesů se dá pro rychlejší vytvoření programu využít toho, že proces je rozdělen na aktivity, které jsou definované rozhraním. Jelikož aktivity mohou být vytvořeny odděleně a poté až spojeny v jeden ucelený celek, tak se na nich může podílet více programátoru. Každý zúčastněny programátor dostane k implementaci jinou aktivitu, či větev procesu, a na základě rozhraní se poté spojí. Pokud máme proces rozdělený na aktivity, tak můžeme využít i toho, že každou aktivitu můžeme testovat zvlášť.

Pro využití WF 4.0 v prostředí distribuovaných systému tato technologie nabízí možnost serializovat celé řešení do XAML řetězce a poslat ho po síti. Díky tomuto převodu můžeme vytvořené řešení uložit do databáze, a poté i dynamicky načítat a spouštět.

Nemyslím si, že by tato technologie vedla k vytváření trhů s workflow komponentami, ale dovedu si představit využití konceptu znovupoužitelnosti softwarových komponent při snadné customizaci již existujících „workflow“ řešení. Takový to přístup může znamenat zkrácení času pro vývoj softwaru a tím poskytnout významnou konkurenční výhodu na trhu. WF 4.0 má jistě své nezastupitelné místo v implementaci bussines pravidel, nicméně na kolik bude úspěšný v konkurenci ostatních produktů, ukáže až čas.

Pro zjednodušení konfigurace WF 4.0 obsahuje Visual Studio designér, pomocí kterého se dá definovat workflow. V TFS je WF 4.0 použito pro konfigurování MSBuild, kde se využívá pro vytváření konfiguračních souborů. Konfigurační soubory obsahují přesně danou strukturu a skládají se z opakujících se bloků. V MSBuildu jsou vytvořeny aktivity, které reprezentují tyto bloky a pomocí definování a složením těchto aktivit vytváříme konfigurační soubor. Vytvořený

konfigurační soubor má grafickou reprezentaci, která slouží k lepší orientaci. Zrychlí se tím vytváření samotného konfiguračního souboru, protože místo složitého vypisování, soubor tvoříme jen přetahováním aktivit z toolboxu, a odstraní se tím i syntaktické chyby spojené s překlepy.

Pokud bychom dokázali, zakomponovat tento designér do vlastní aplikace, byl by to velký přínos. Mohli bychom vytvořit aplikaci, která by obsahovala designér pro tvorbu samotného procesu. Poté by nám stačilo vytvořit sadu aktivit zaměřených na určitý proces a přidat je do aplikace. Zákazník, který by měl takovou aplikaci, by si mohl definovat procesy pro své zákazníky sám, s pomocí designéru a sady aktivit. Vytvářet vlastní procesy pomocí této technologie by určitě nemohl vykonávat každý, ale jen zaškolený personál, ale i tak by to byl velký přínos pro firmy. Tato práce by mohla být rozšířena o pokus vytvořit aplikaci, která by v sobě obsahovala takovýto designér a zmapovat všechny její možné výhody a nevýhody.

7 Literatura

[1] Václav Řepa: *Vývojové trendy metodik vývoje informačních systémů – výzva BPR.* Vysoká škola ekonomická, Katedra informačních technologií, nám. W.Churchilla 4, Praha 1999

[2] Robert L. Glass: *Loyal Opposition: Is There Really a Software Crisis?* IEEE Software 15(1): 104-105 (1998)

[3] Clemens Szyperski: *Component Software – Beyond Object – Oriented Programming – Second Edition*, Addison-Wesley/AVM Press, 2002 ISBN 0-201-17888-5

[4] Aalst, W., Hee, K.: *Workflow Management: Models, Methods and Systems*. The MIT Press, London, 2004. 384 s. ISBN 978-0262720465.

[5] Martin Mayer: *Informační podpora procesů krizového řízení.* Masarykova Univerzita, Brno 2010

[6] Prof. Ing. Ivo Vondrák, CSc: *Metody byznys modelování.* VŠB-TU Ostrava, Ostrava 2004

[7] Doc. RNDr. Jaroslav Markl: *Petriho sítě 1.* VŠB-TU Ostrava,

[8] Kenn Scribner: *Windows Workflow Fondation.* Microsoft Press,Washington 2007

[9] Andrew Zhu: *Microsoft Windows Workflow Foundation 4.0 CookBook.* Packt Publishing, Birmingham 2010

[10] Mark J. Collins: *Windows Workflow in .NET 4.0.*Apress, New York 2010

[11] Daniel Albrecht: *Nové trendy ve workflow.* Dokument dostupný na URL http://is.muni.cz/th/140438/fi_m/DP.pdf (Květen 2012)